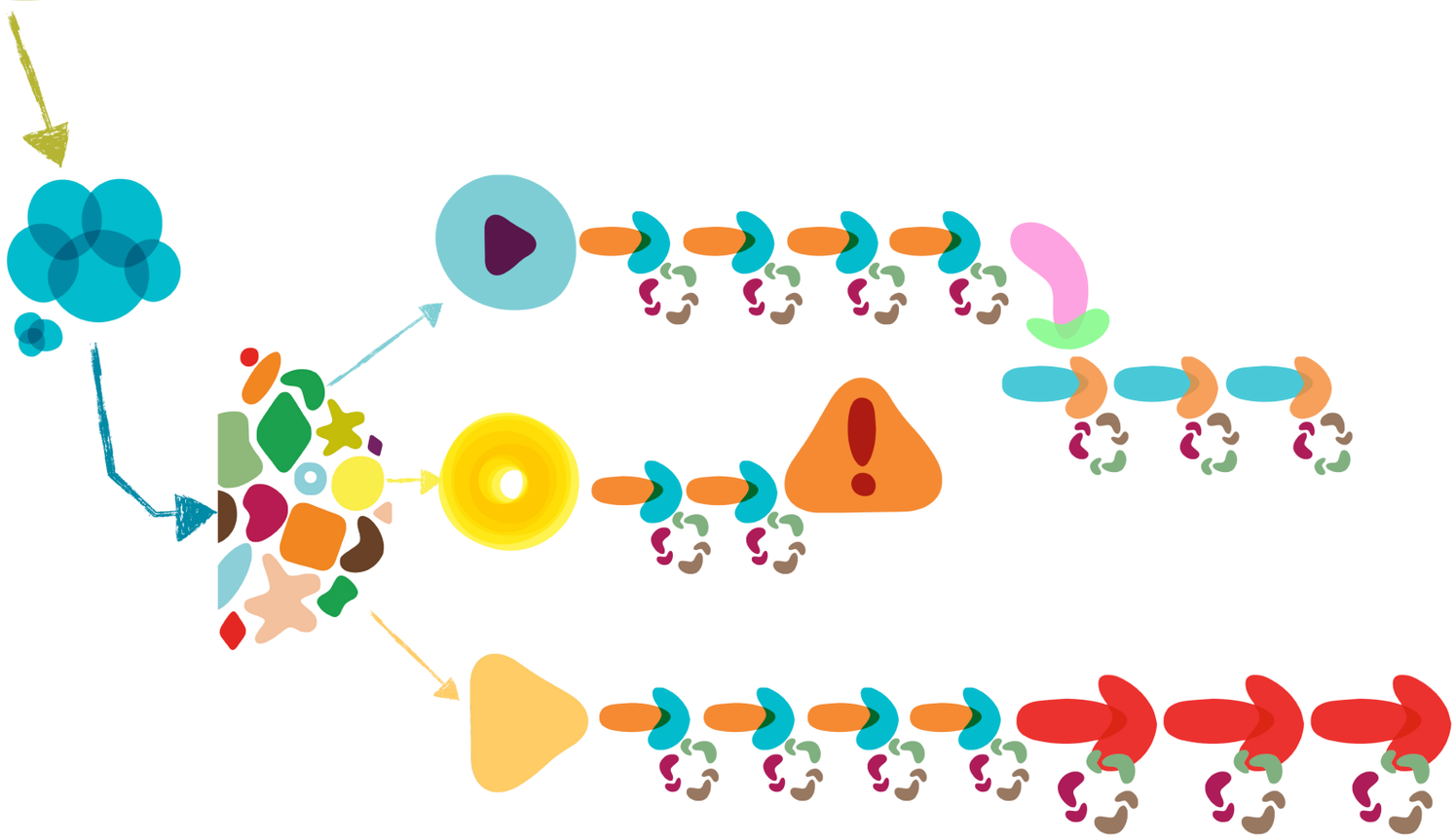
Continuous Delivery for Architects

6 Ways Continuous Delivery Impacts Architects



ThoughtWorks®

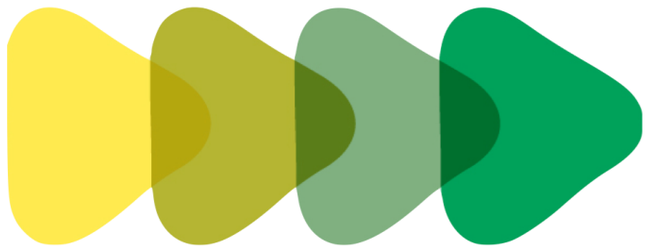
NEAL FORD

Director / Software Architect / Meme Wrangler

@neal4d

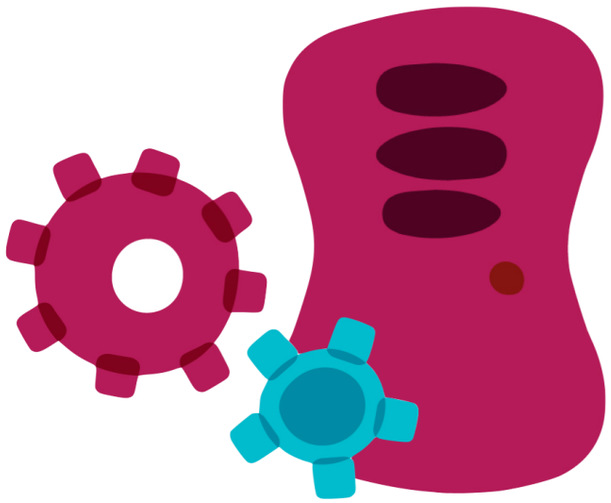
nealford.com

Continuous Delivery



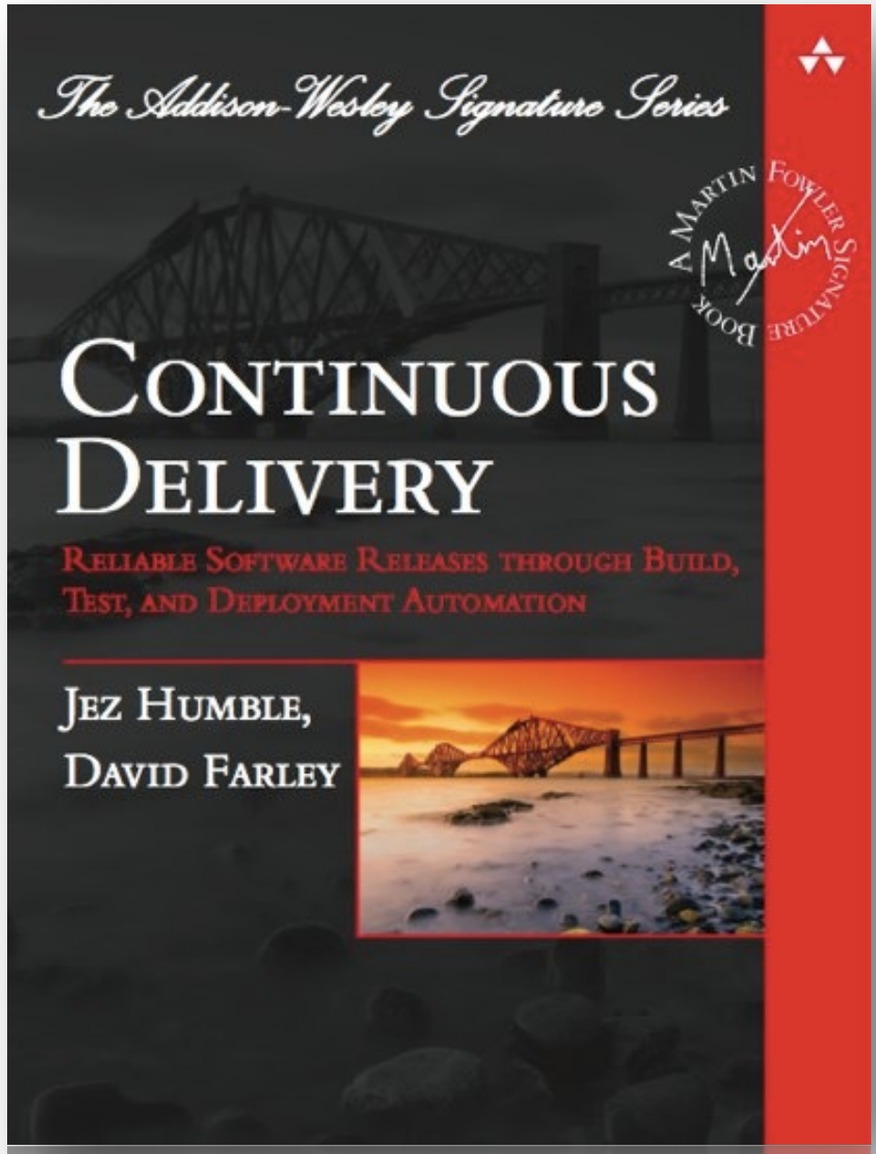
deployment pipelines

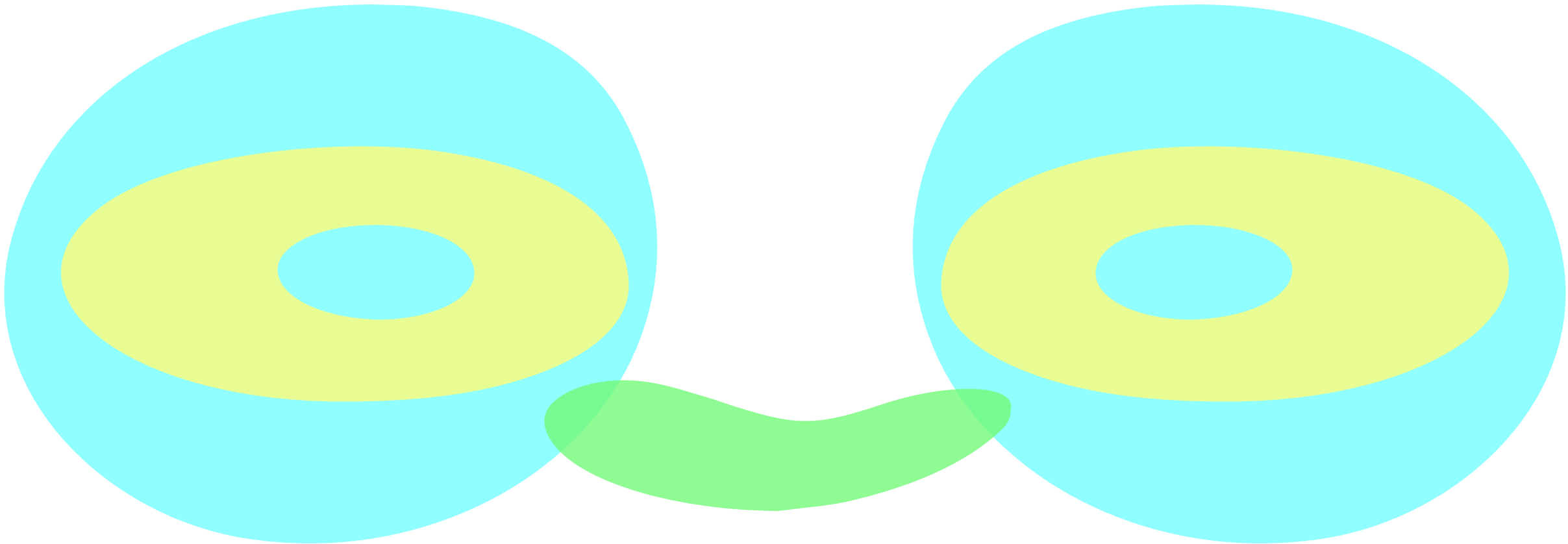
tests, synergistic practices,
incremental deployment

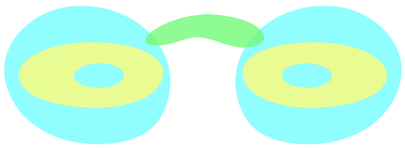


data & infrastructure

Effective engineering practices
for software projects.







Yesterday's best practice

is tomorrow's anti-pattern.

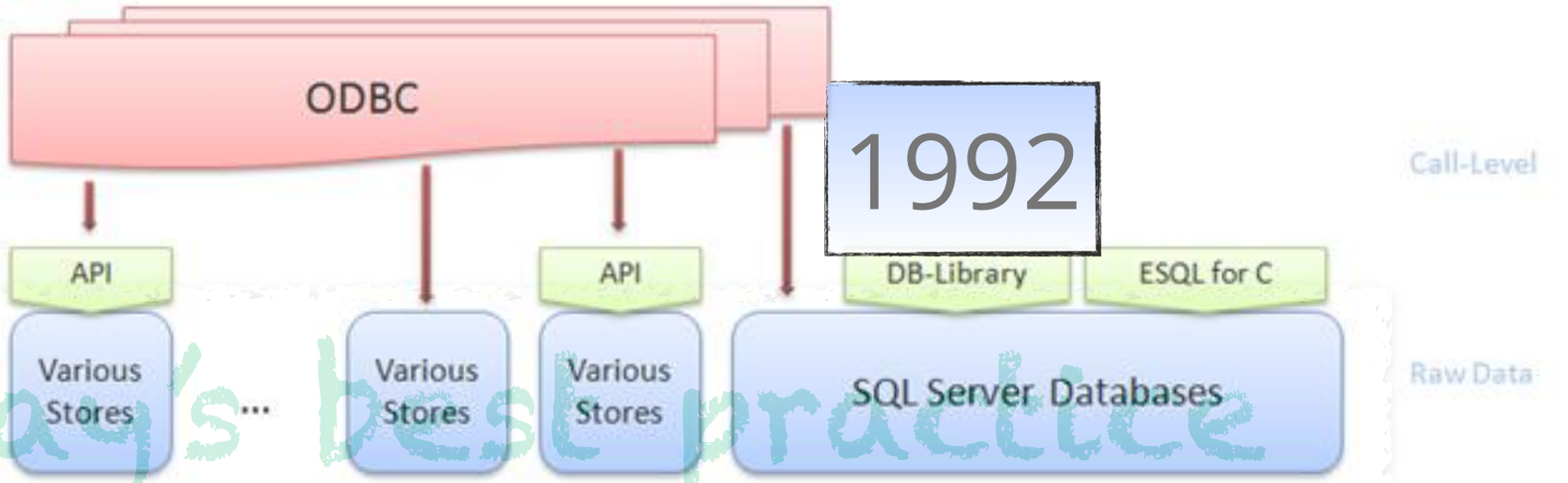
A Case against the GO TO Statement.

by Edsger W.Dijkstra
Technological University
Eindhoven, The Netherlands

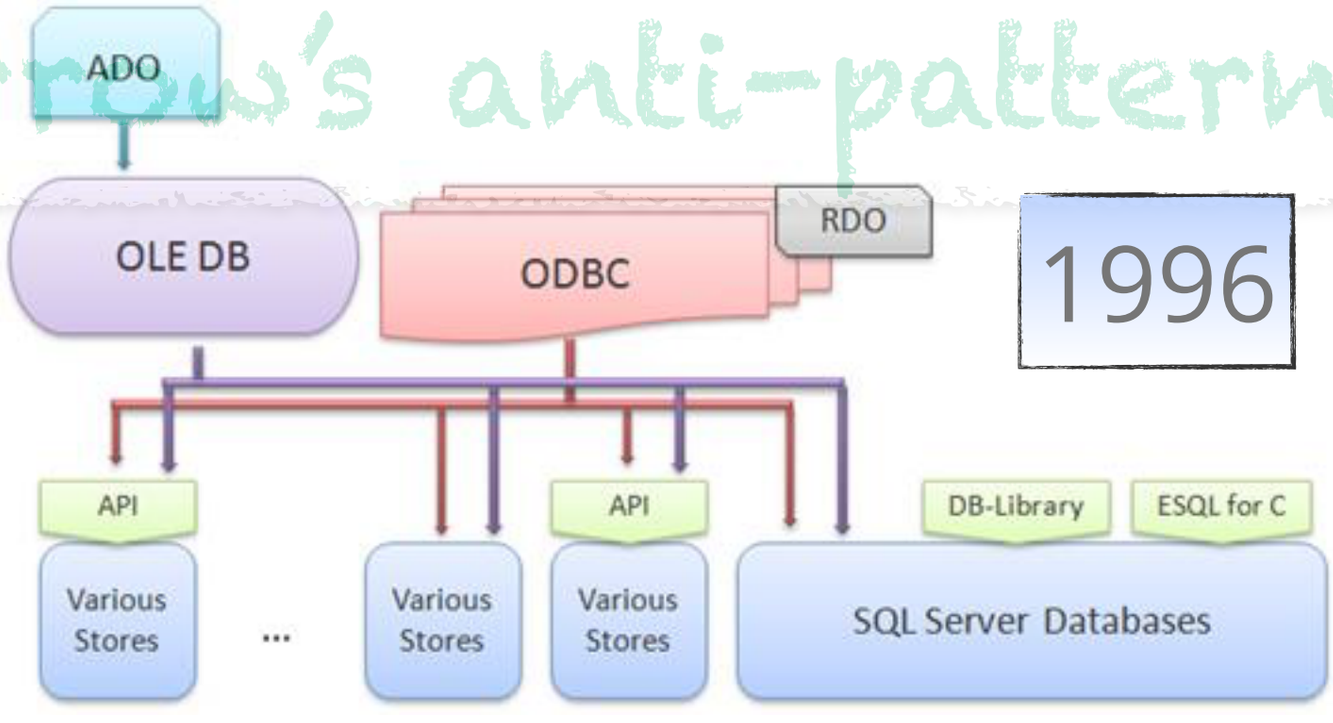
Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except -perhaps- plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

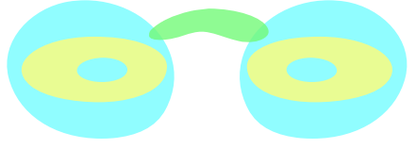
My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to effectuate the desired effect; it is this process that in its dynamic behaviour has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF

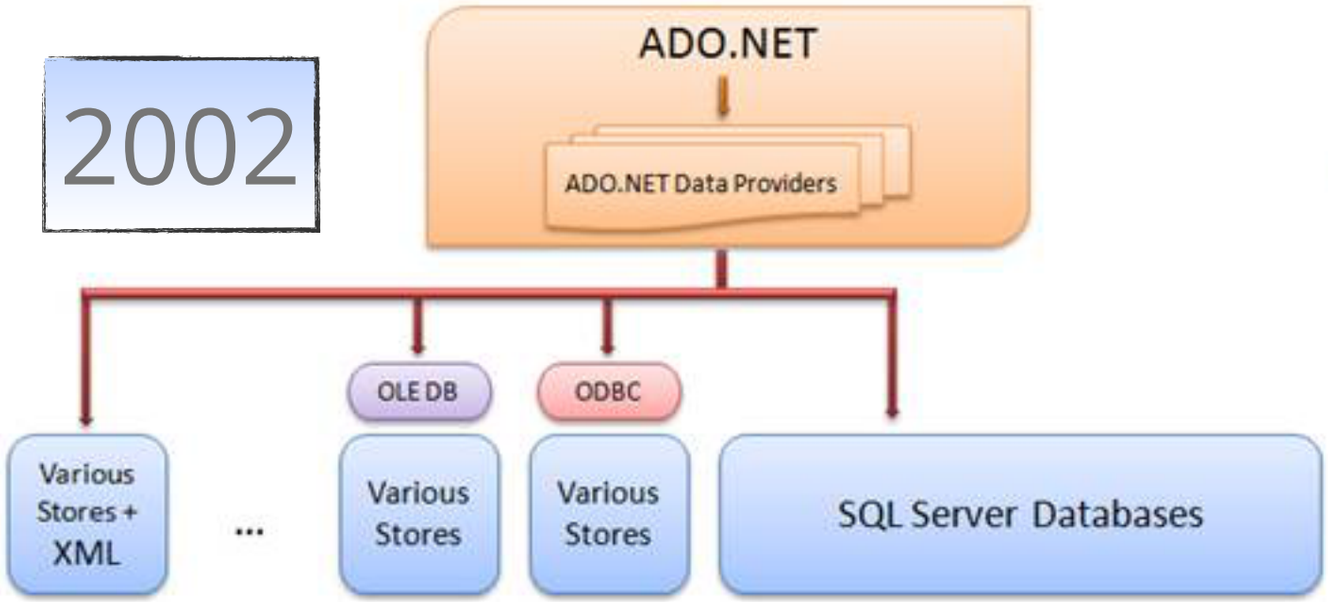


Yesterday's best practice is tomorrow's anti-pattern.





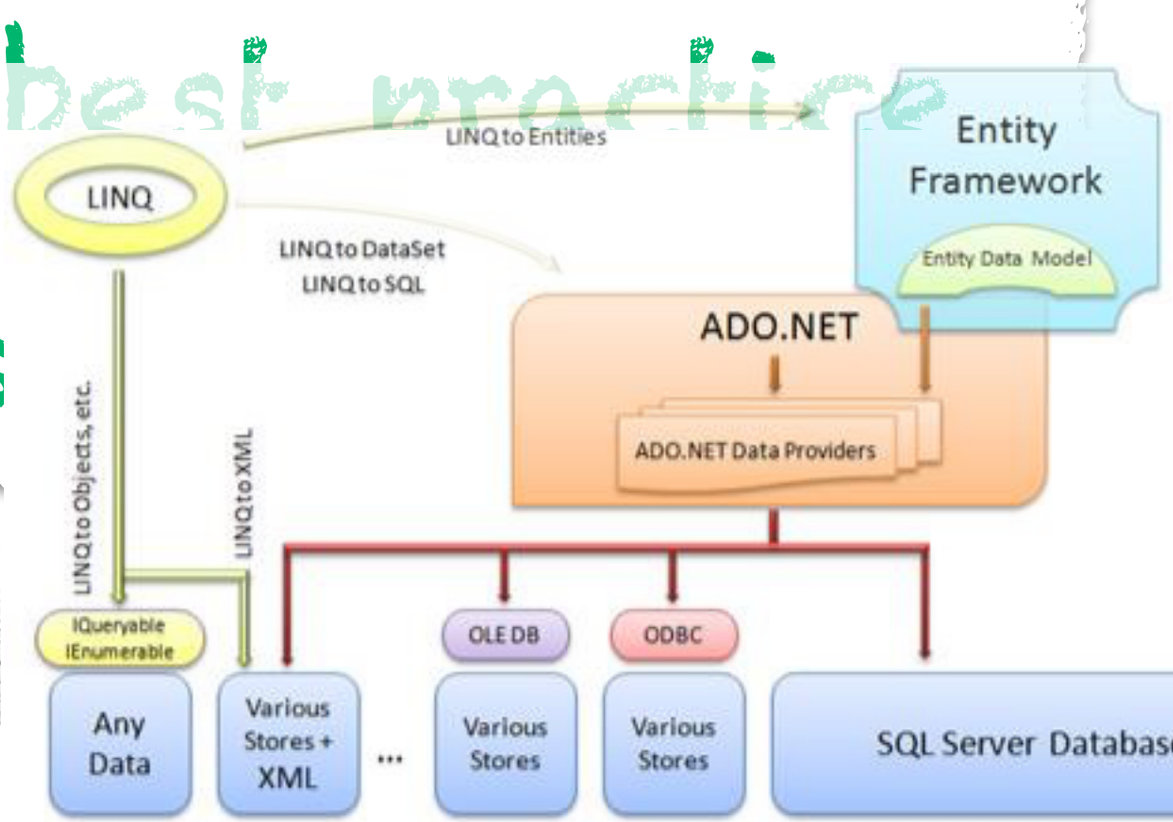
2002

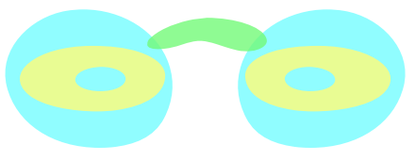


Yesterday's best practice is tomorrow's

2007

2008

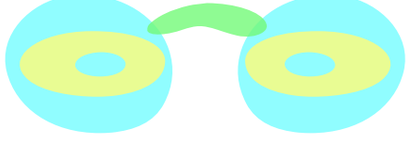




Yesterday's best practice

is tomorrow's anti-pattern.

ENTERPRISE
Java Beans

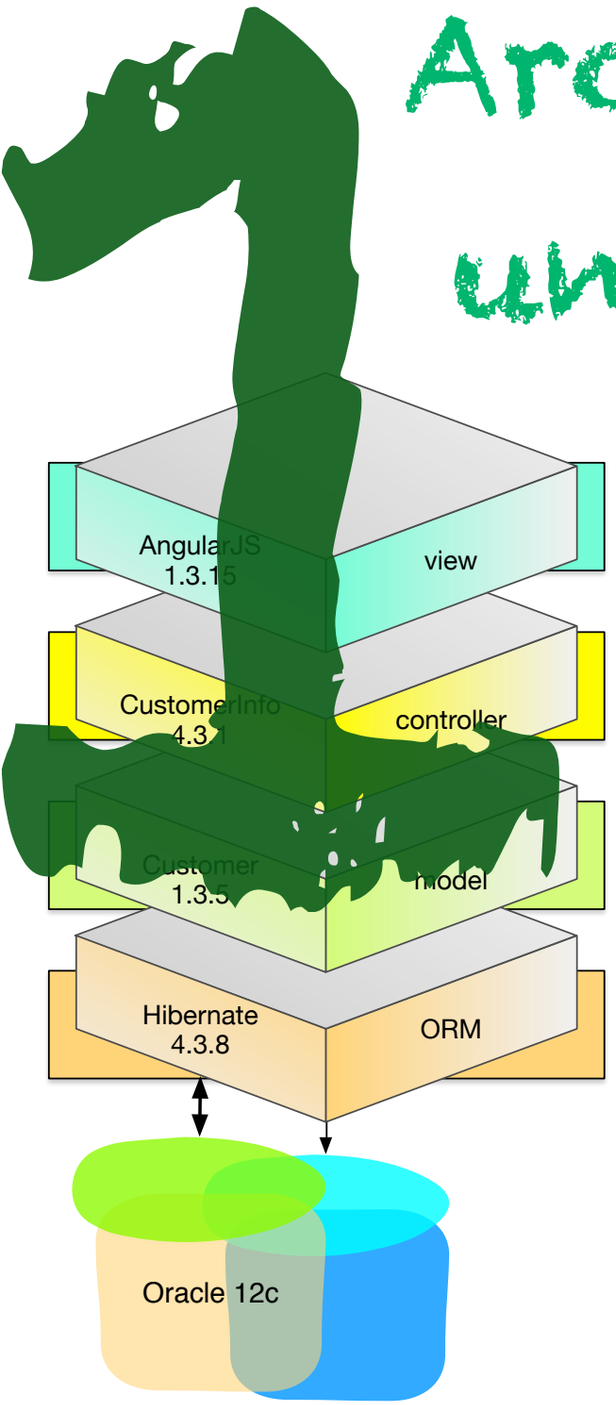


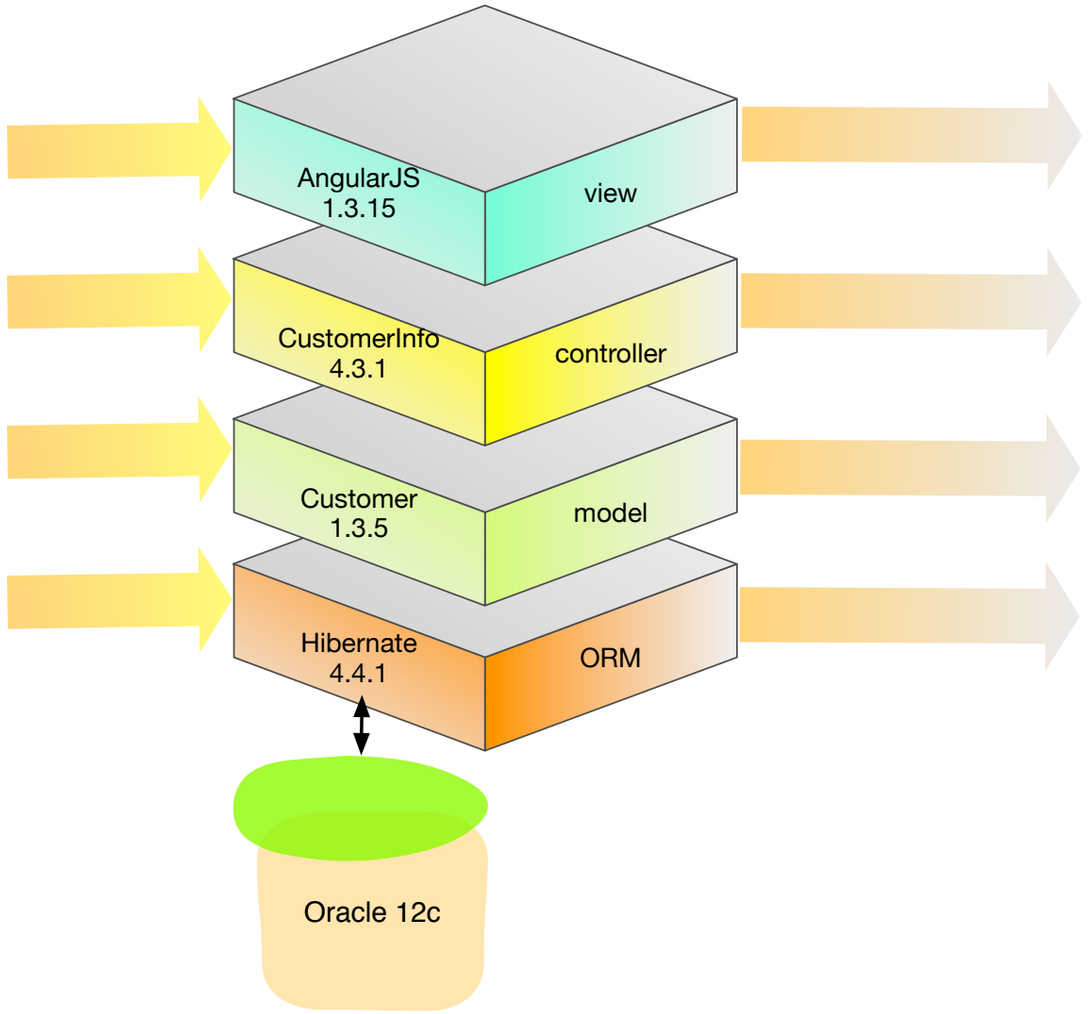
martinfowler.com/bliki/AntiPattern.html

Architecture is abstract until operationalized.

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html

Architecture is abstract until operationalized.

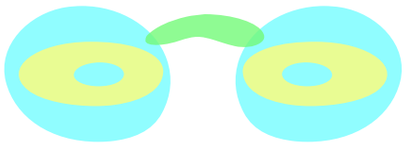


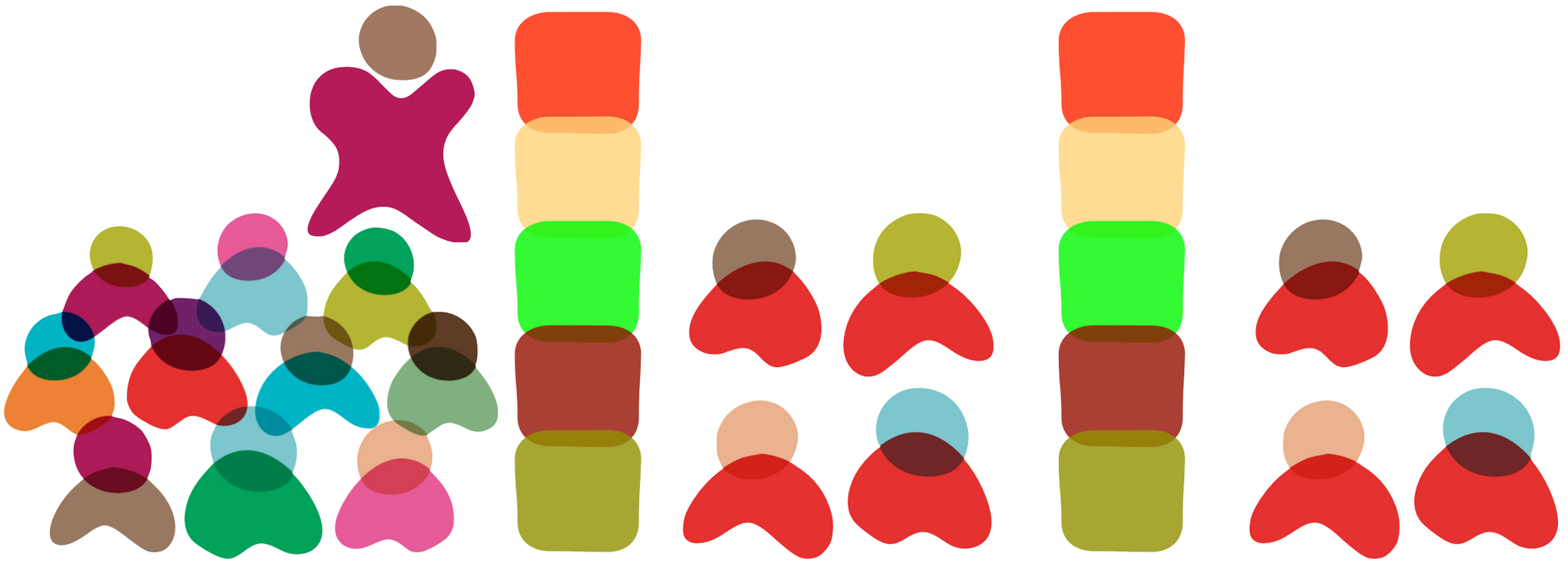


2D 3D 4D

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html

Expanding Role of Architect

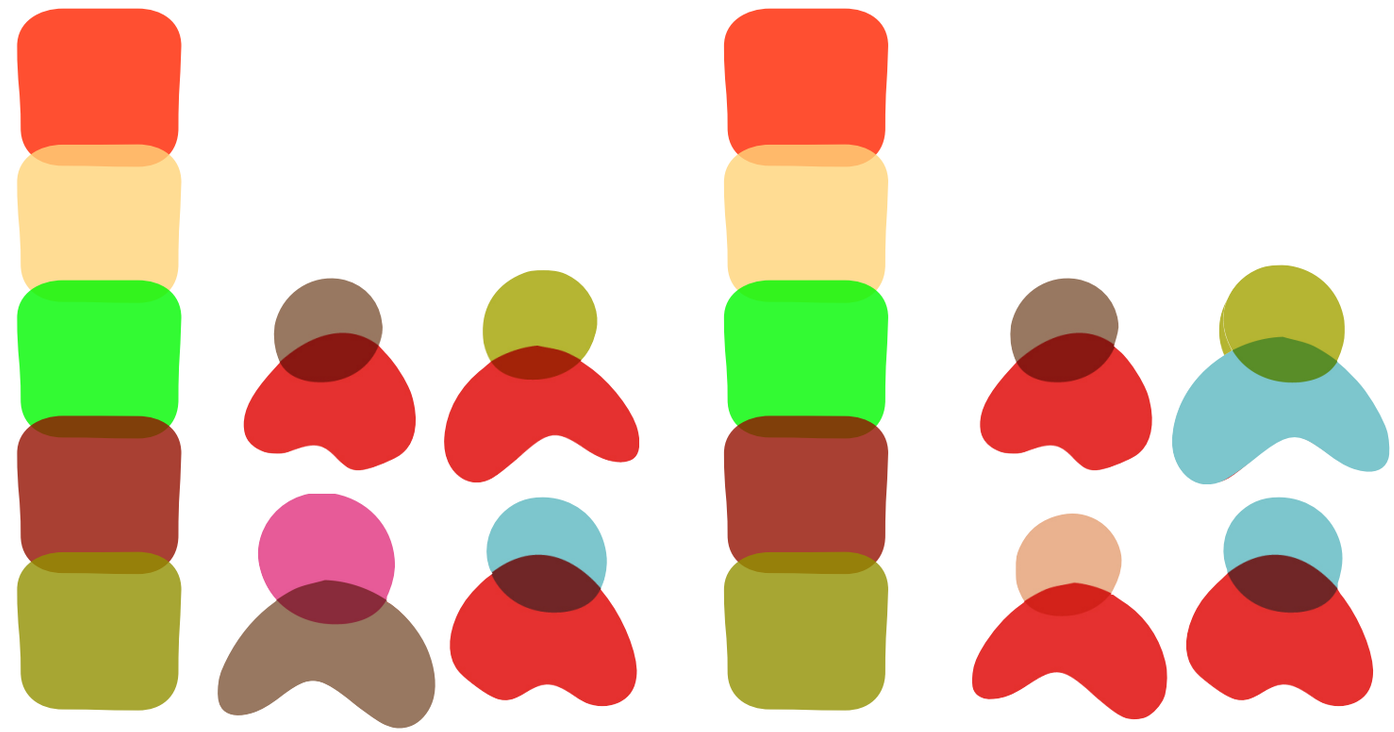




engineering DBAs operations

pre-CD

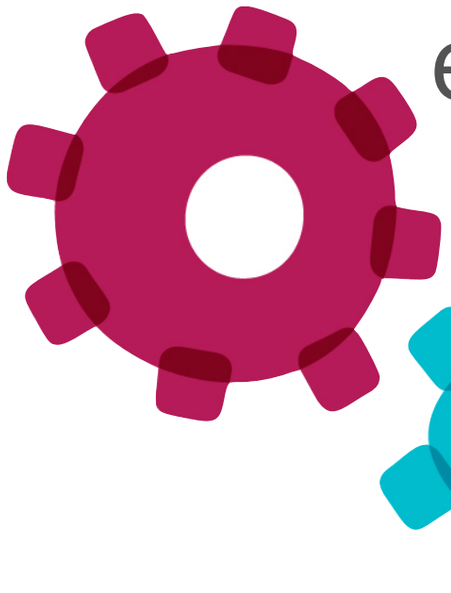
Expanding Role of Architect



engineering DBAs operations

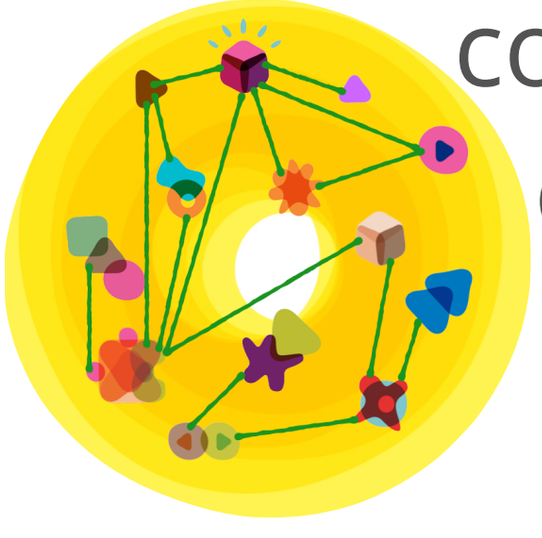
post-CD

Agenda



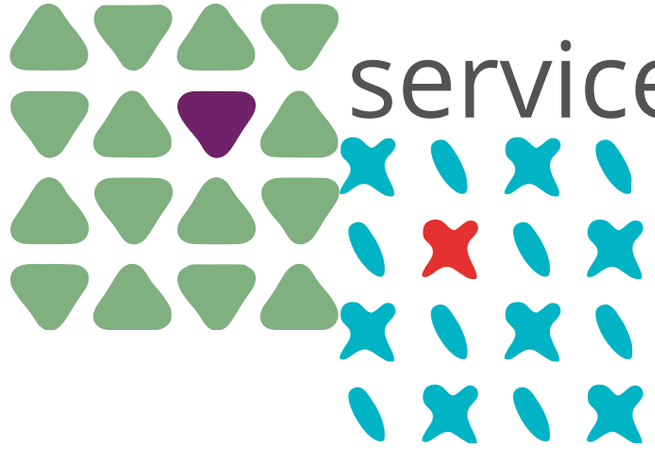
engineering practices

deployment models

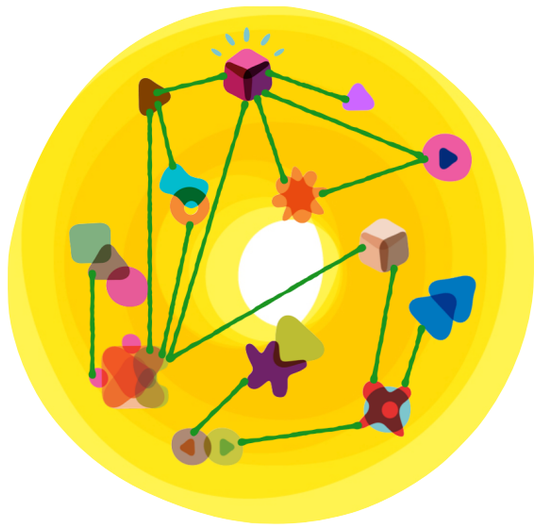


coupling & cohesion

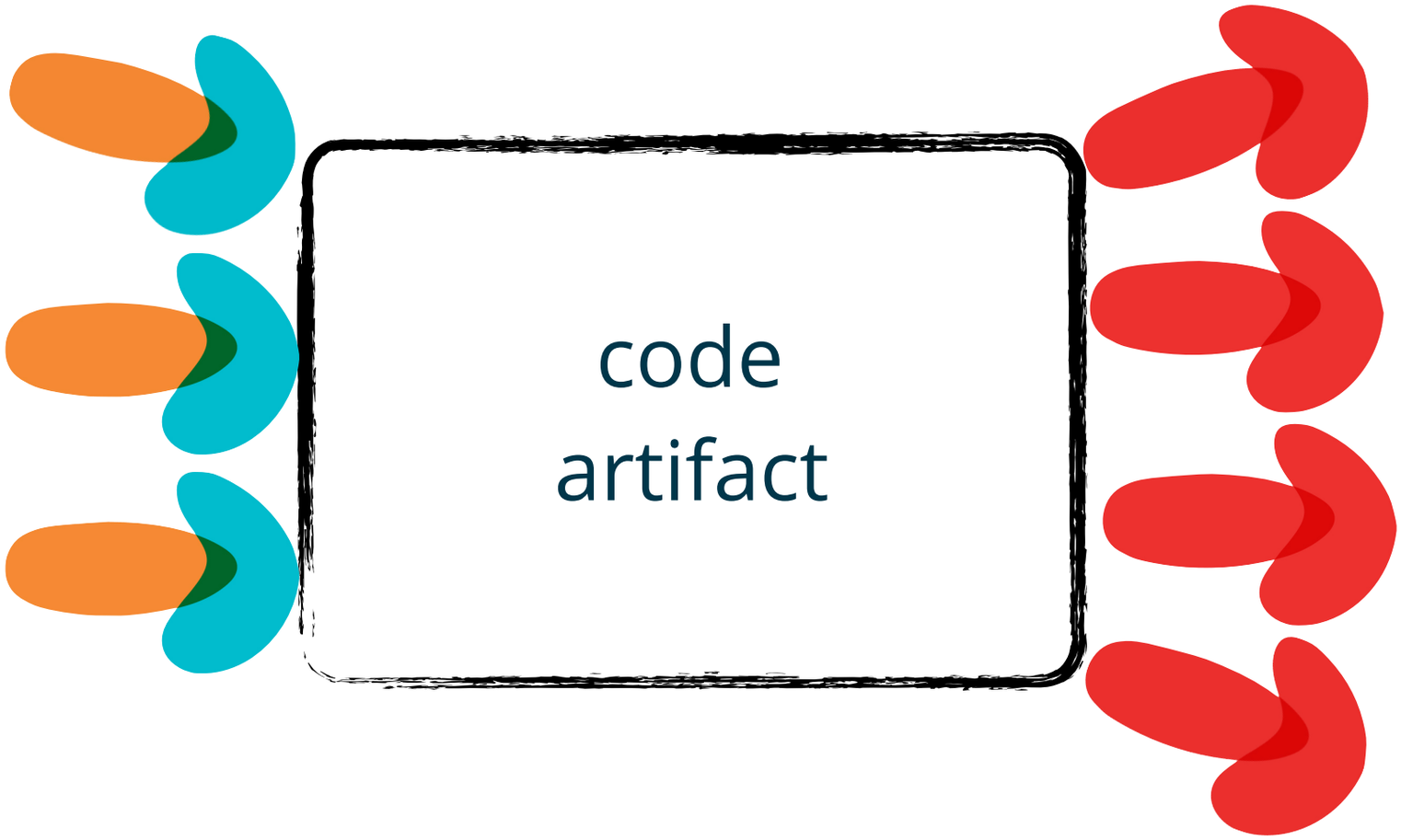
components & services



Metrics & Visualizations



Structural Coupling Metrics



afferent

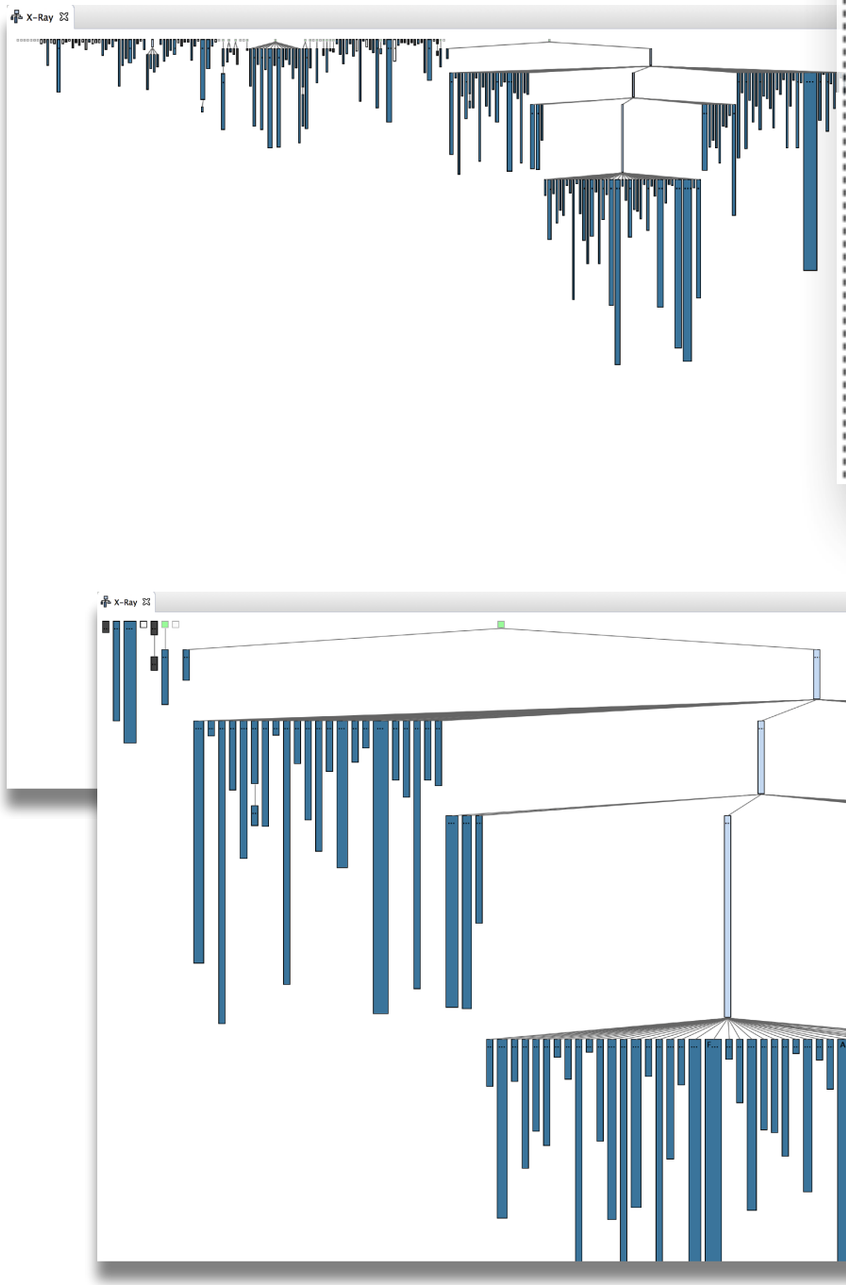
efferent

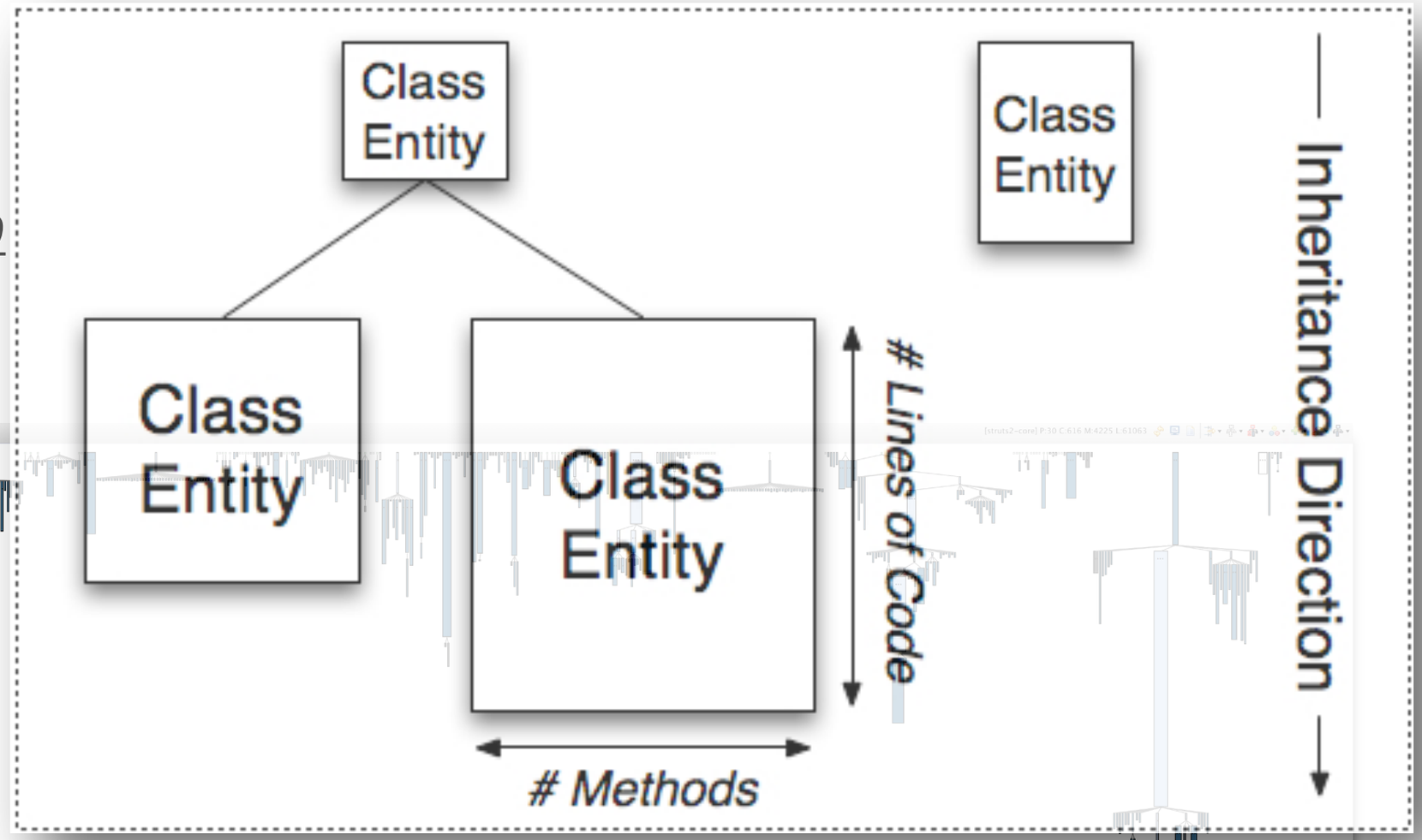
exit

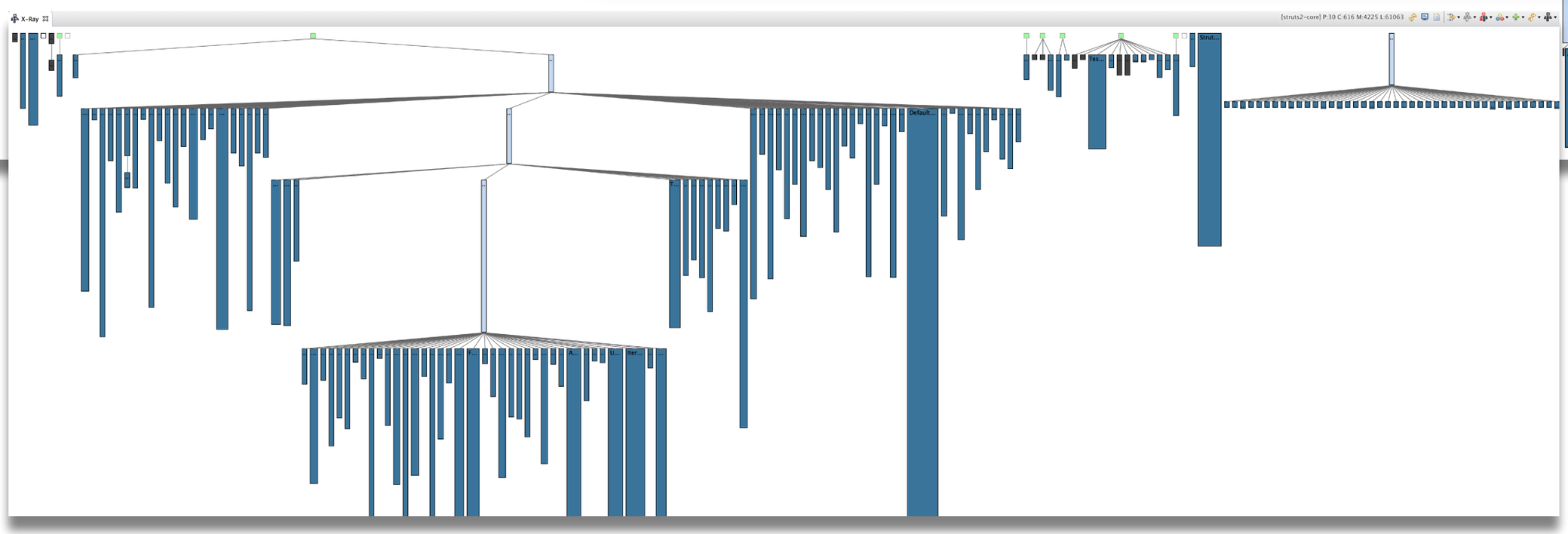
Temporal Coupling

X-Ray

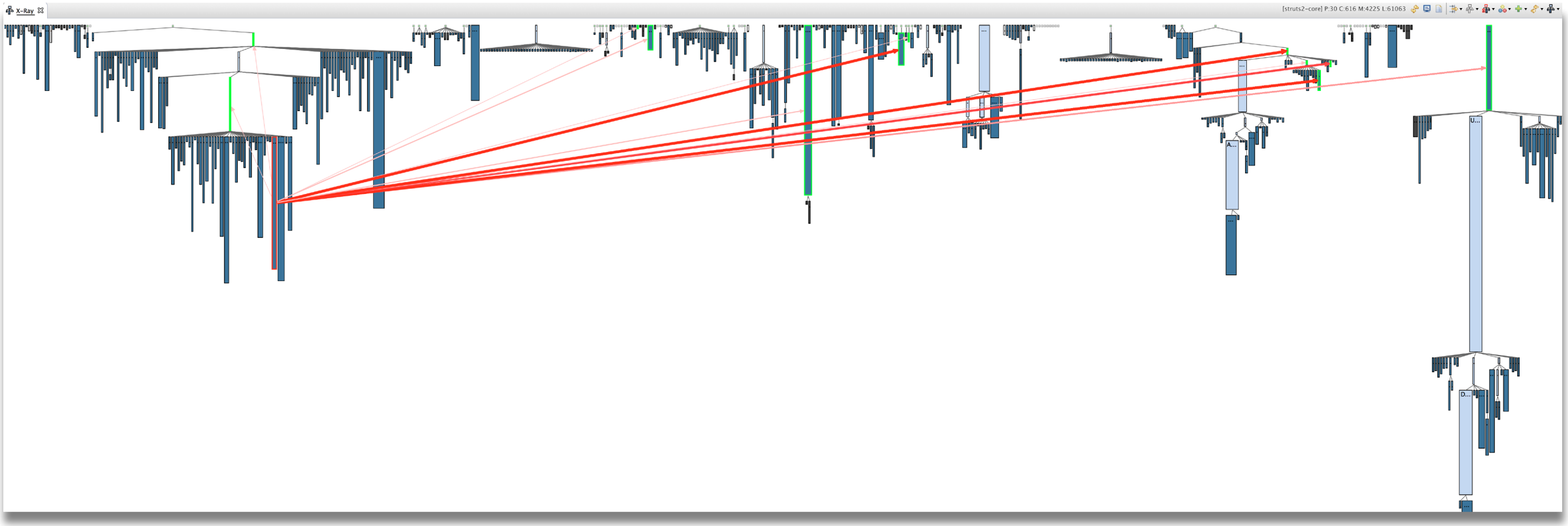
xray.inf.usi.ch/xray.php

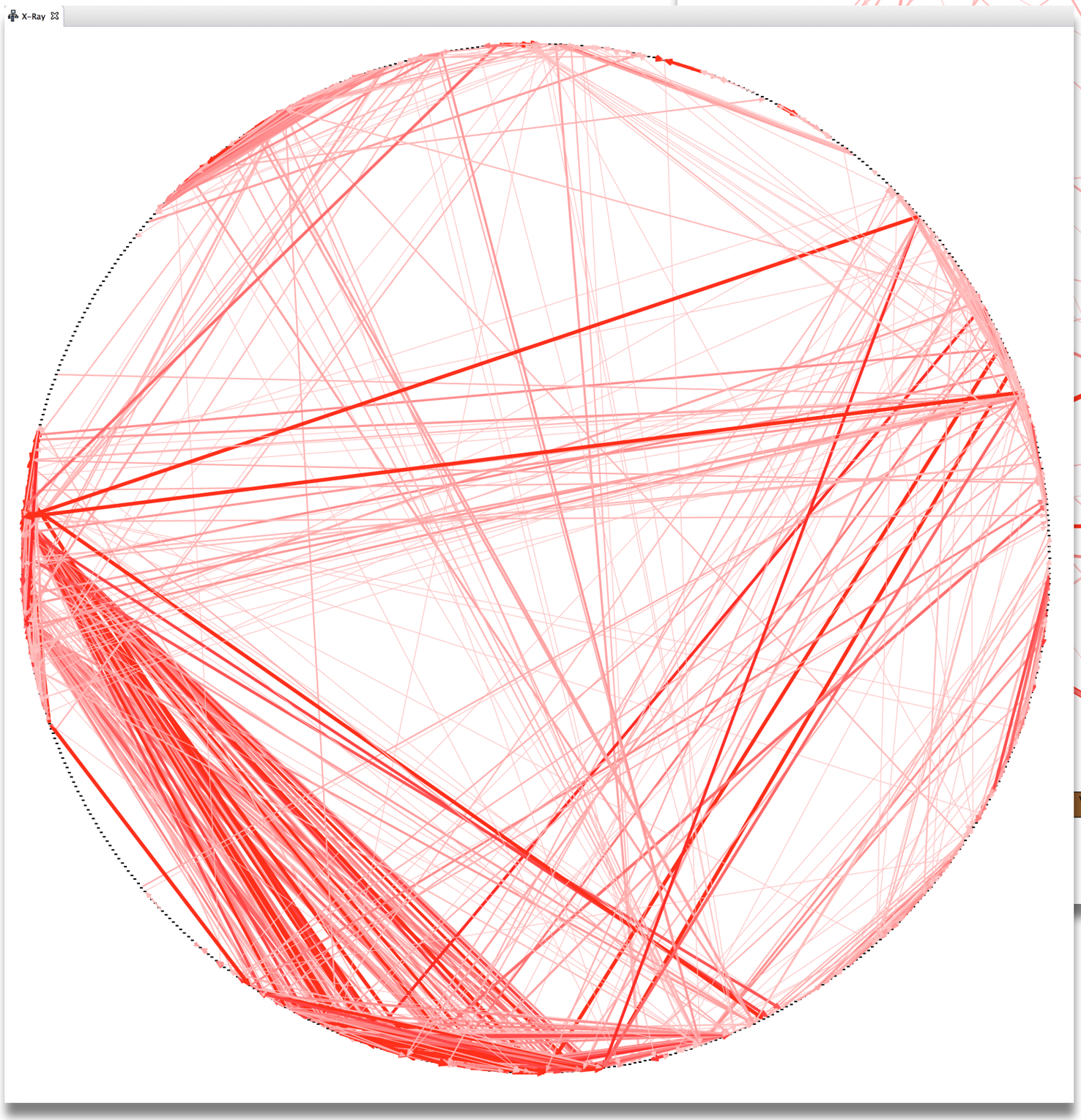


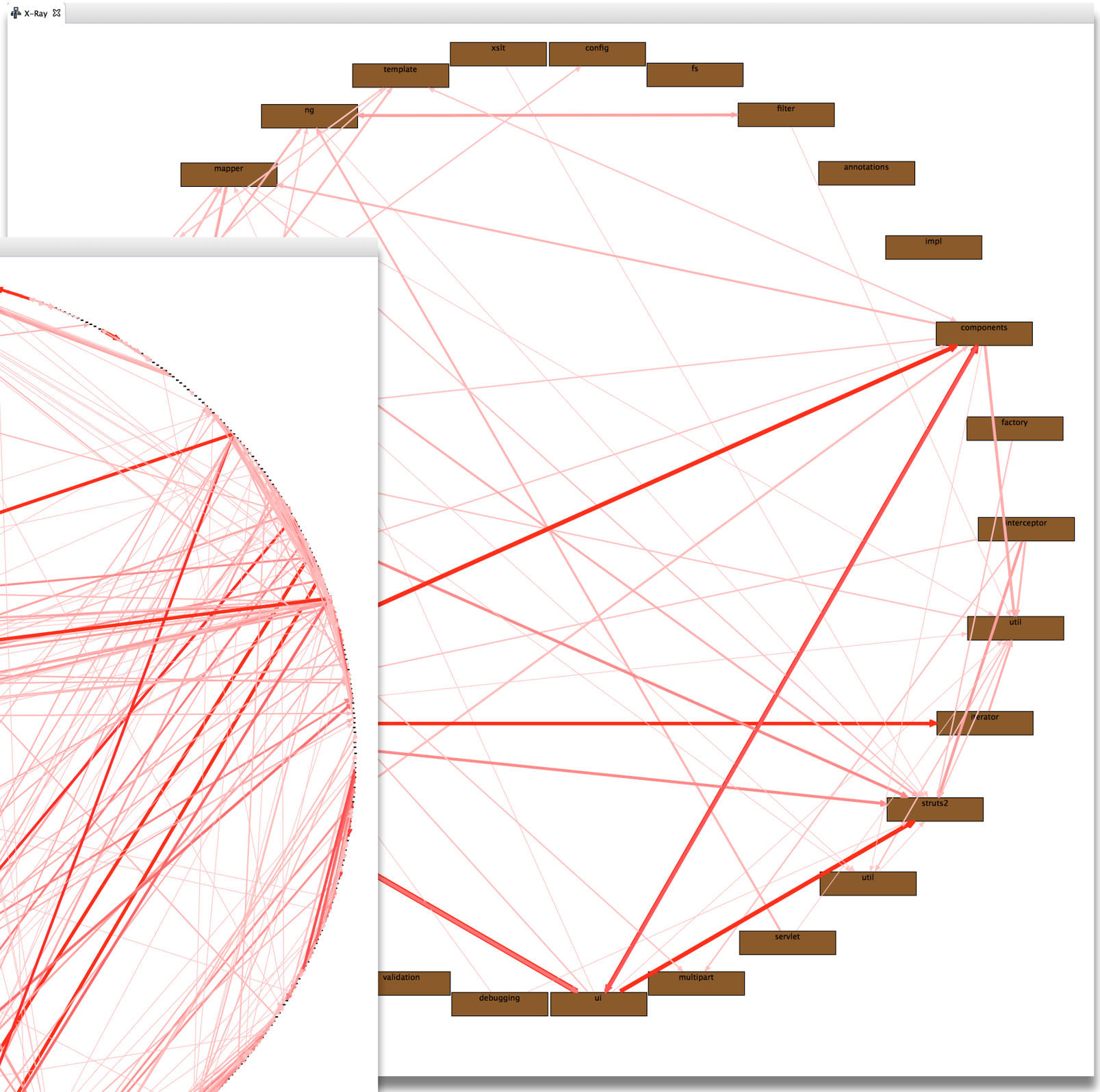




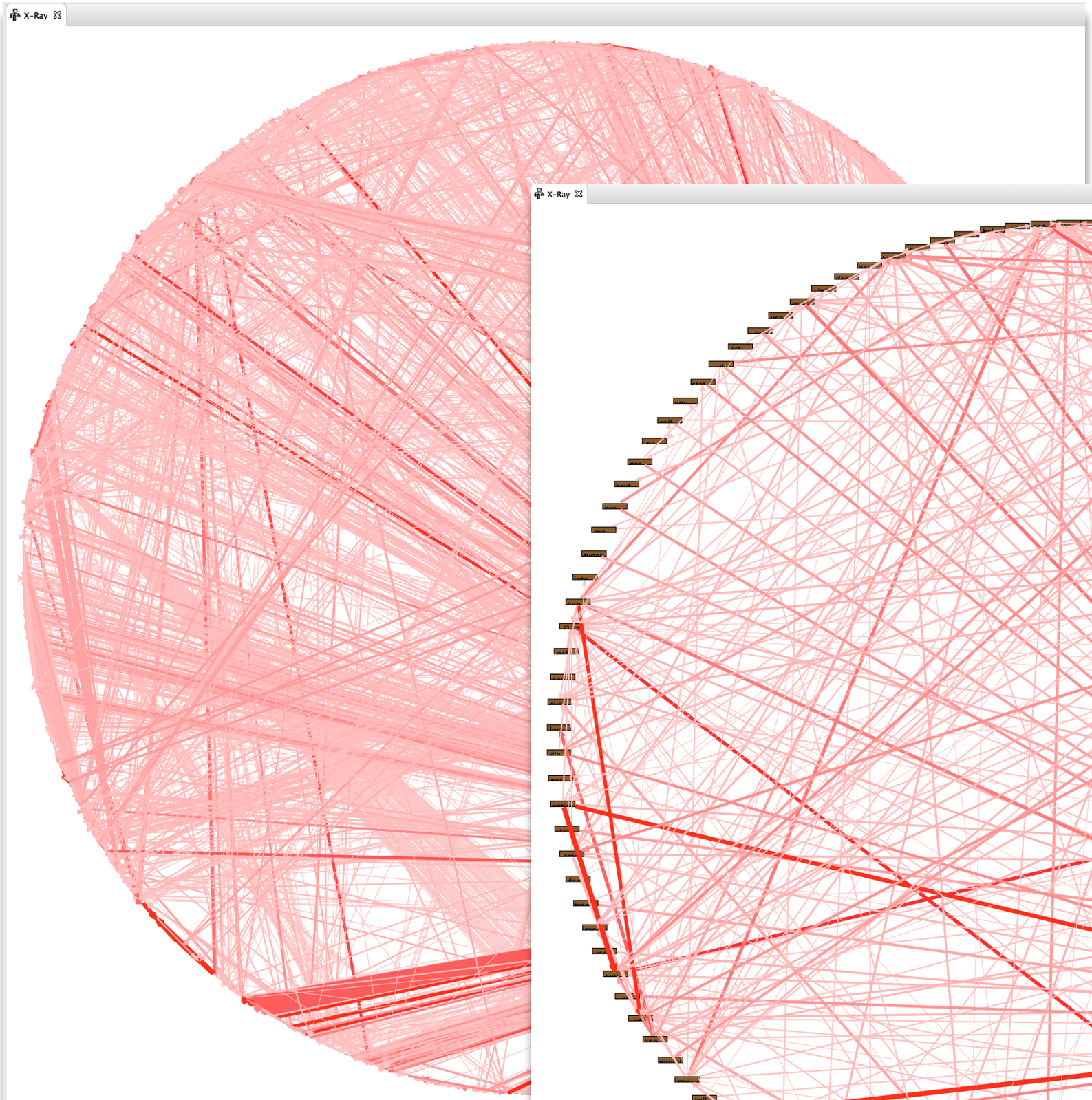
XRay Dependencies

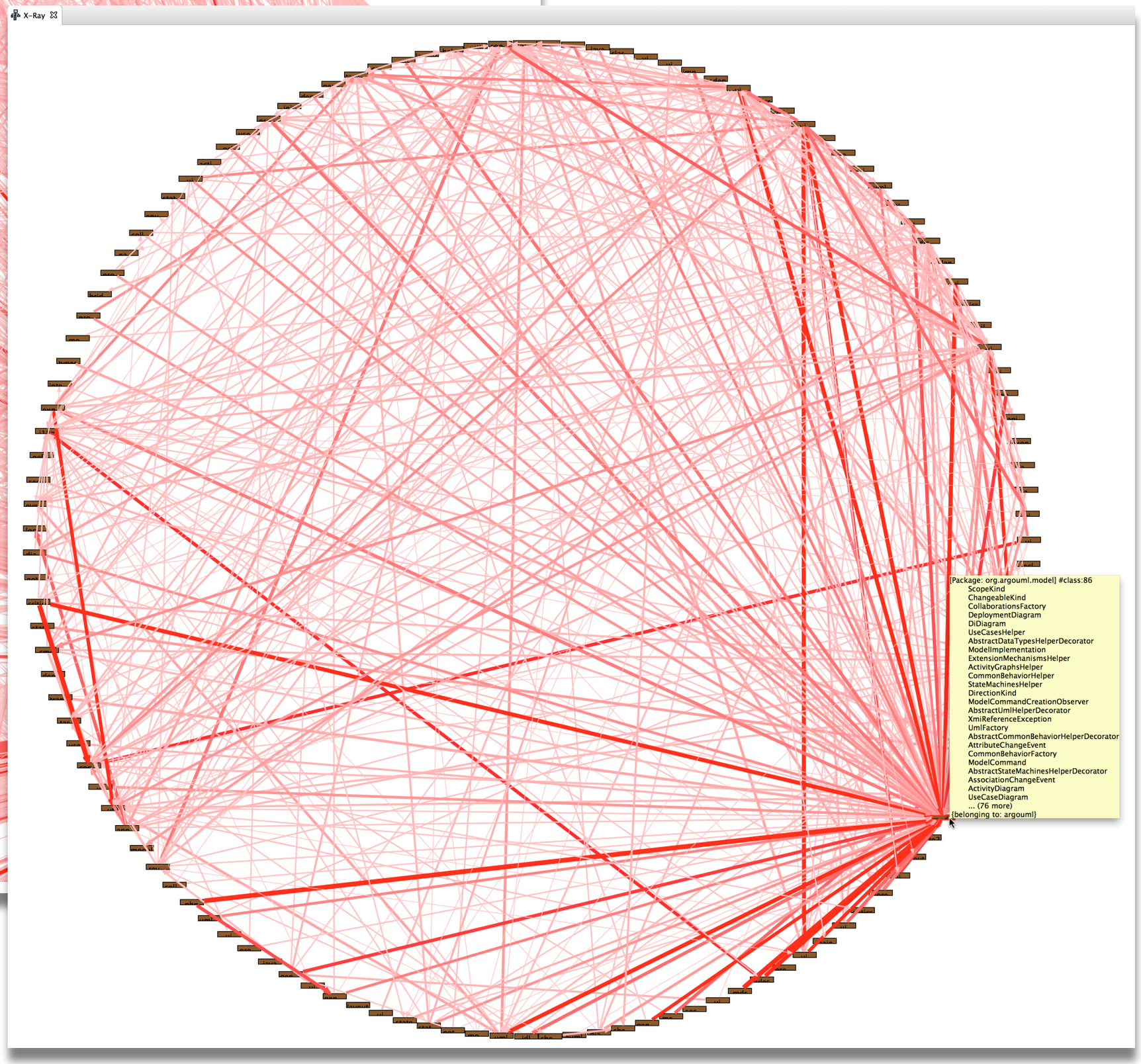






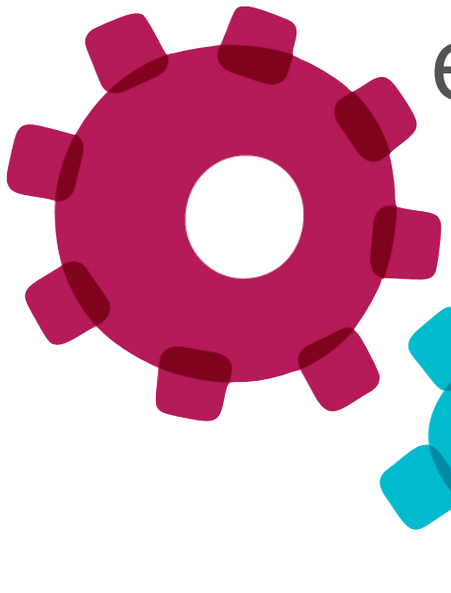
Class/package Dependencies


X-Ray


X-Ray
[Package: org.argouml.model] #class:86
ScopeKind
ChangeableKind
CollaborationsFactory
DeploymentDiagram
DiDiagram
UseCasesHelper
AbstractDataTypesHelperDecorator
ModelImplementation
ExtensionMechanismsHelper
ActivityGraphsHelper
CommonBehaviorHelper
StateMachinesHelper
DirectionKind
ModelCommandCreationObserver
AbstractUmlHelperDecorator
XmiReferenceException
UmlFactory
AbstractCommonBehaviorHelperDecorator
AttributeChangeEvent
CommonBehaviorFactory
ModelCommand
AbstractStateMachinesHelperDecorator
AssociationChangeEvent
ActivityDiagram
UseCaseDiagram
... (76 more)
{belonging to: argouml}

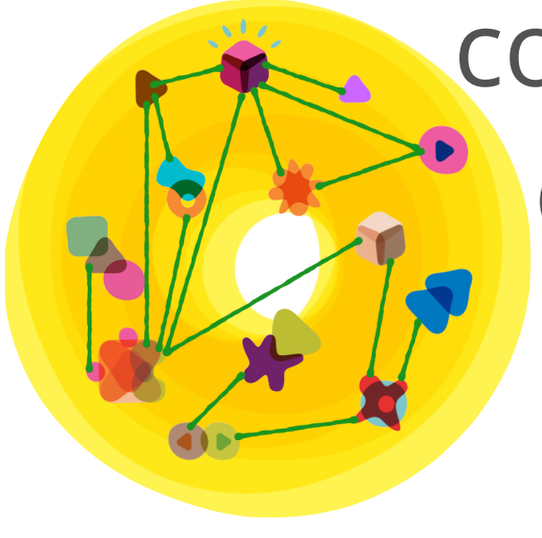
Understand the structure of your code as it evolves.

Agenda



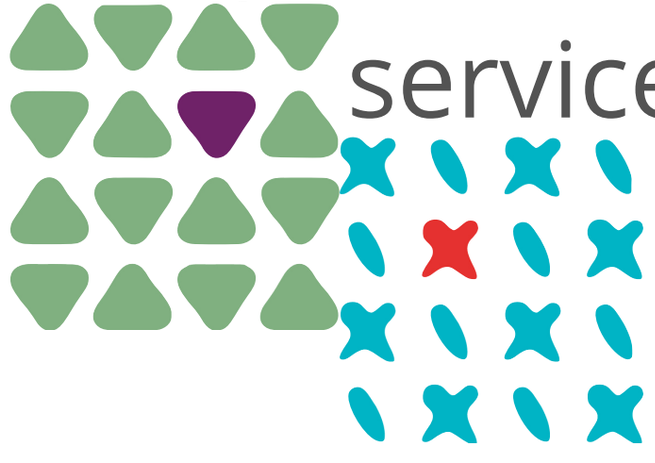
engineering practices

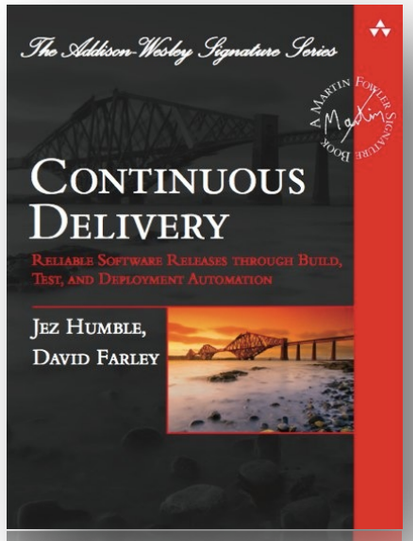
deployment models



coupling & cohesion

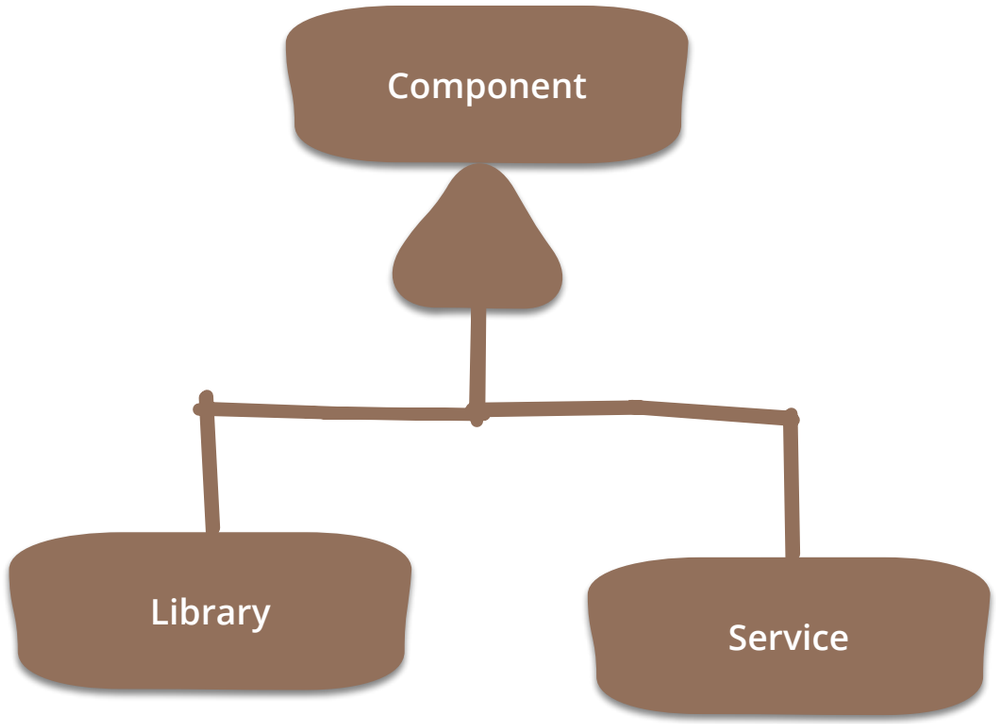
components & services

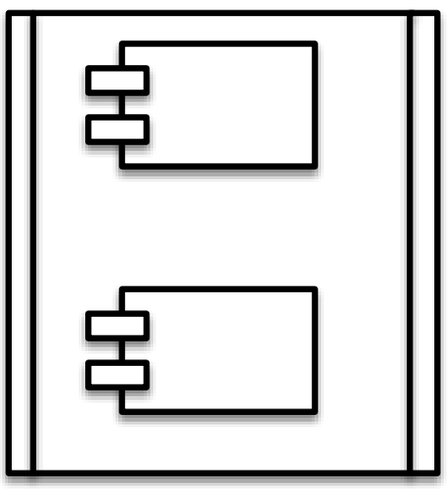




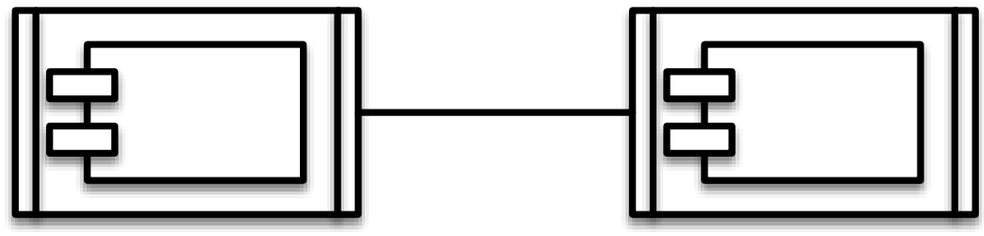
“Use components to decouple parts of your application that change at different rates.”

Components are units of software that can be independently replaced and upgraded



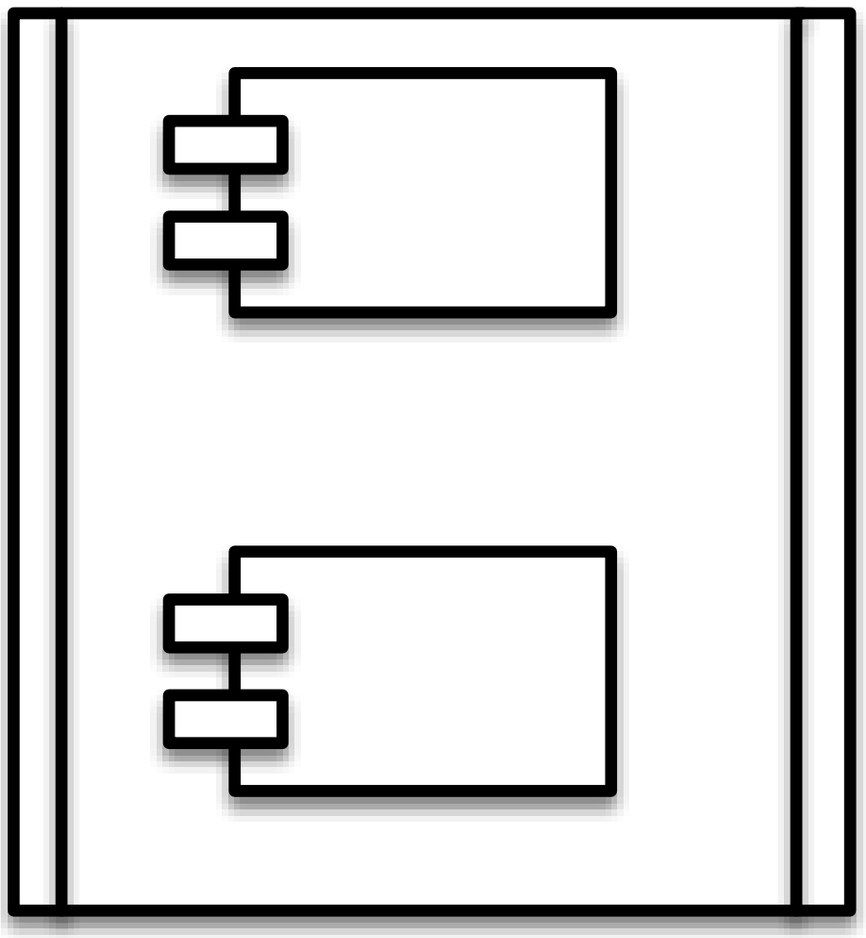


Libraries run within a single process, communicating through language function call mechanisms

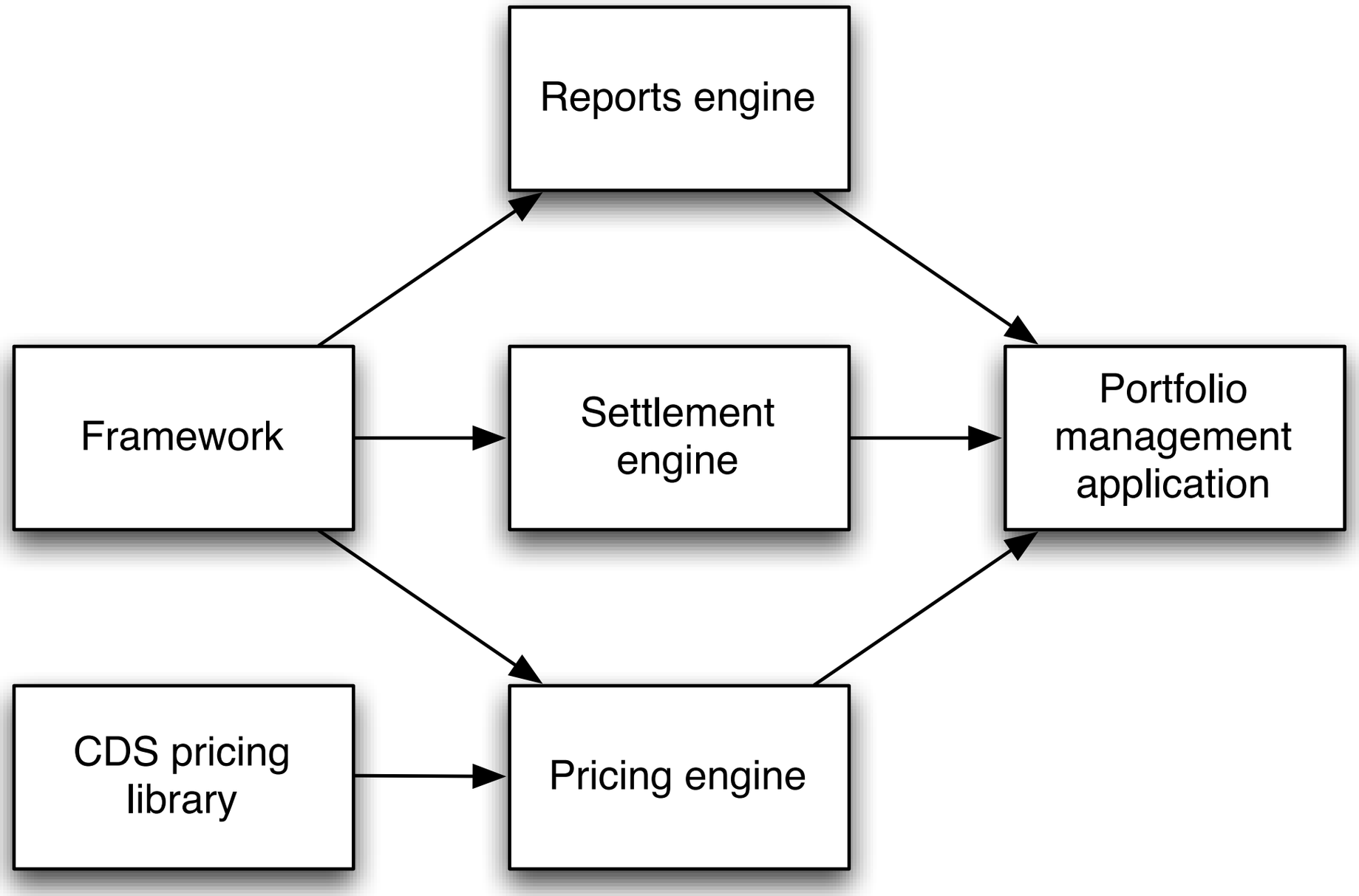


Services run in separate processes, communicating with networking mechanisms such as HTTP or TCP/IP

Components



Libraries


Reports engine
Framework
Settlement engine
Portfolio management application
CDS pricing library
Pricing engine

Managing Dependencies

"DLL Hell"

2 approaches:

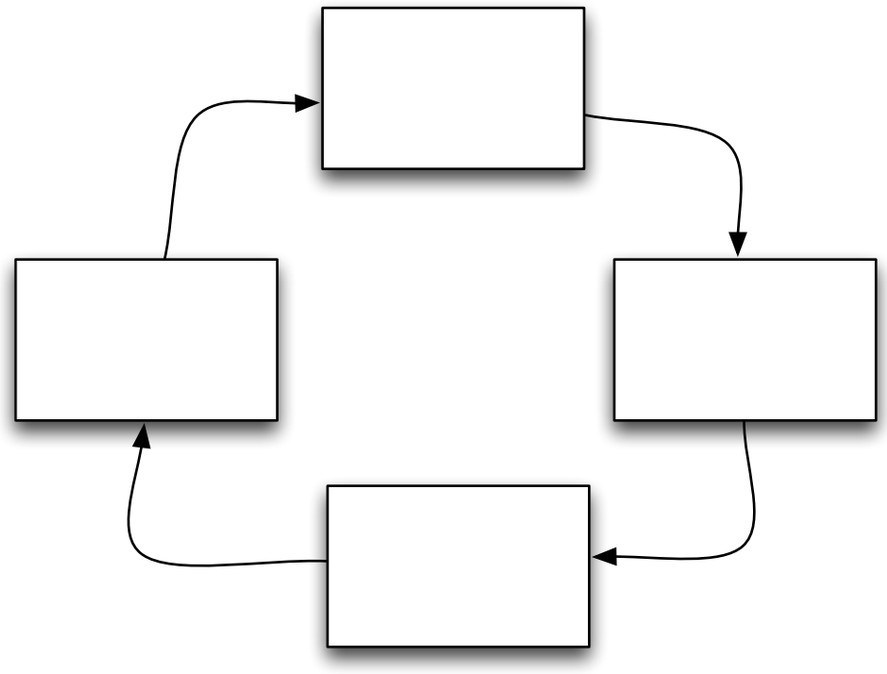
the ubiquitous lib directory

transitive dependency management tool*

*all platforms (eventually) rely on tooling

Anti-pattern: Cycles

circular dependencies



Java tooling promotes/ignores cycles

makes the system hard to componentize

makes lifecycle more complex

JarAnalyzer Analysis

Run with JarAnalyzer on

Summary

[summary] [jars] [cycles] [explanations]

Jar Name	Total Classes	Abstract Classes	Packages	Level	Abstractness	Efferent	Afferent	Instability	Distance
antlr.jar	210	48	10	1	0.23	0	1	0.00	0.77
commons-beanutils.jar	66	7	4	2	0.11	2	3	0.40	0.49
commons-collections.jar	187	15	3	1	0.08	0	4	0.00	0.92
commons-digester.jar	55	9	3	3	0.16	3	2	0.60	0.24
commons-fileupload.jar	16	4	1	1	0.25	0	1	0.00	0.75
commons-logging.jar	18	2	2	1	0.11	0	4	0.00	0.89
commons-validator.jar	30	1	2	4	0.03	5	1	0.83	0.14
jakarta-oro.jar	62	13	6	1	0.21	0	1	0.00	0.79
struts.jar	289	33	25	5	0.11	7	0	1.00	0.11

Jars

[summary] [jars] [cycles] [explanations]

antlr.jar

Level: 1 Afferent Couplings: 1 Efferent Couplings: 0 Abstractness: 0.23 Instability: 0.00 Distance: 0.77

Uses Jars	Used by Jars	Cycles With
None	struts.jar	*None*

Packages within jar	Unresolved Packages
antlr antlr.build antlr.collections antlr.debug antlr.preprocessor antlr.actions.cpp antlr.actions.csharp antlr.actions.java antlr.collections.impl antlr.debug.misc	*None*

commons-beanutils.jar

Level: 2 Afferent Couplings: 3 Efferent Couplings: 2 Abstractness: 0.11 Instability: 0.40 Distance: 0.49

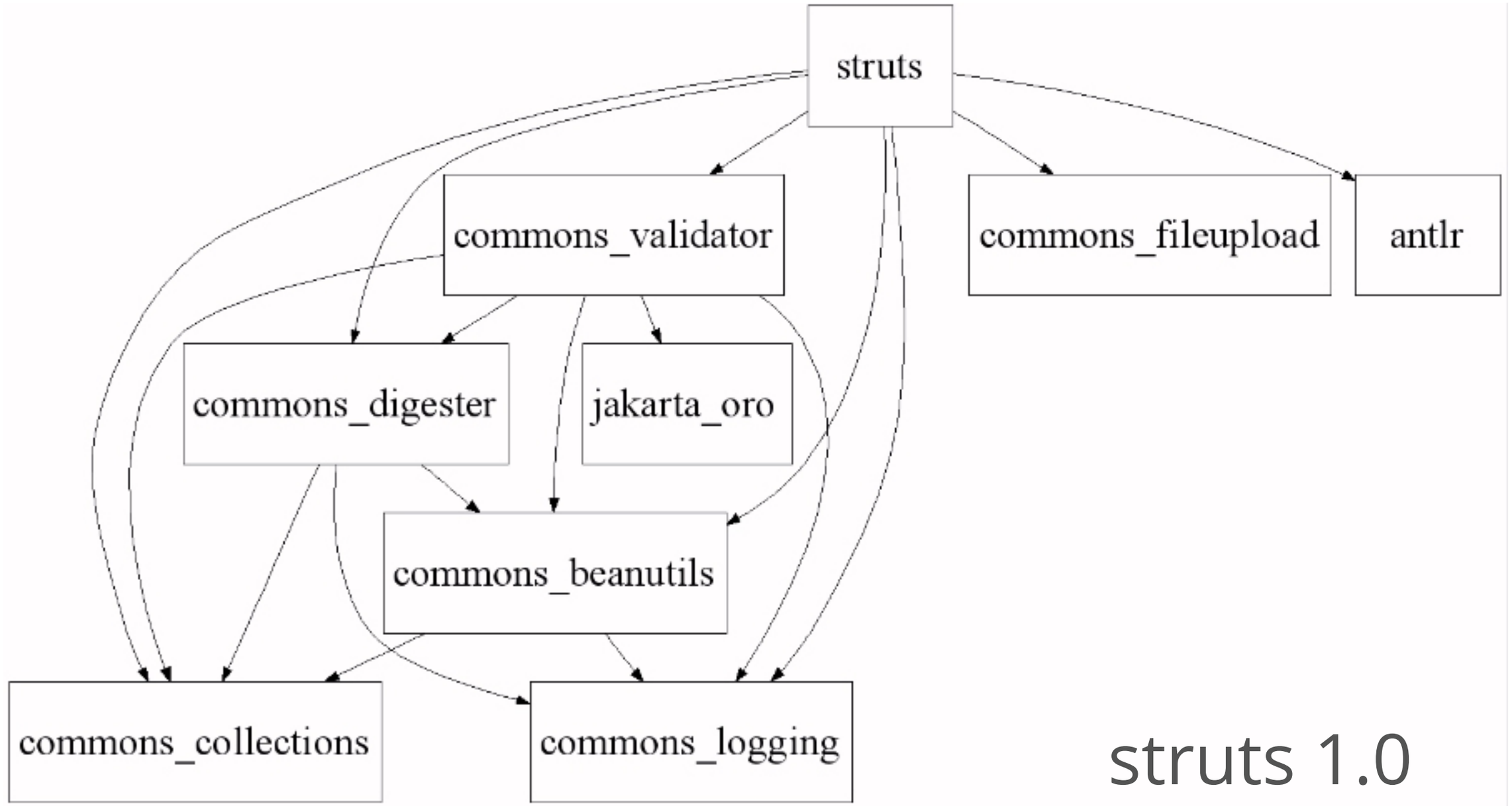
Uses Jars	Used by Jars	Cycles With
commons-collections.jar commons-logging.jar	commons-digester.jar commons-validator.jar struts.jar	*None*

Packages within jar	Unresolved Packages
org.apache.commons.beanutils.converters org.apache.commons.beanutils org.apache.commons.beanutils.locale.converters org.apache.commons.beanutils.locale	*None*

Kirk Knoernschild

http://www.kirkk.com/main/Main/JarAnalyzer

Consequences of Ignoring...



struts 1.0


struts2_tiles_plugin_2_0_11
struts2_pell_multipart_plugin_2_0_11
lpha_10
_1_1
plexus_utils_1_2
tiles_core_2_0_4
tiles_api_2_0_4
jasper_compiler_5_5_12
jasper_runtime_5_5_12
jasper_compiler_jdt_5_5_12
commons_el_1_0
jdtcore_3_1_0
testng_5_8_jdk15

struts 2


s2_config_browser_plugin_2_0_11
struts2_jfreechart_plugin_2_0_11
struts2_jasperreports_plugin_2_0_11
struts2_sitegraph_plugin_2
struts2_spring_plugin_2_0_11
struts2_core_2_0_11
struts2_plexus_plugin_2_0_11
dwr_1_1_beta_3
xwork_2_0_4
spring_web_1_2_8
ognl_2_6_11
spring_mock_1_2_8
spring_context_1_2_8
antlr_2_7_2
commons_fileupload_1_1_1
spring_aop_1_2_8
commons_io_1_0
spring_beans_1_2_8
spring_core_1_2_8
org_
commons_logging_1_0_4
junit_3_8_1

Structure 101

structure101.com/

analyzes codebase

provides “to do” list of refactorings

allows developers to untangle cycles

clarkware.com/software/JDepend.html

Dependency Cycle

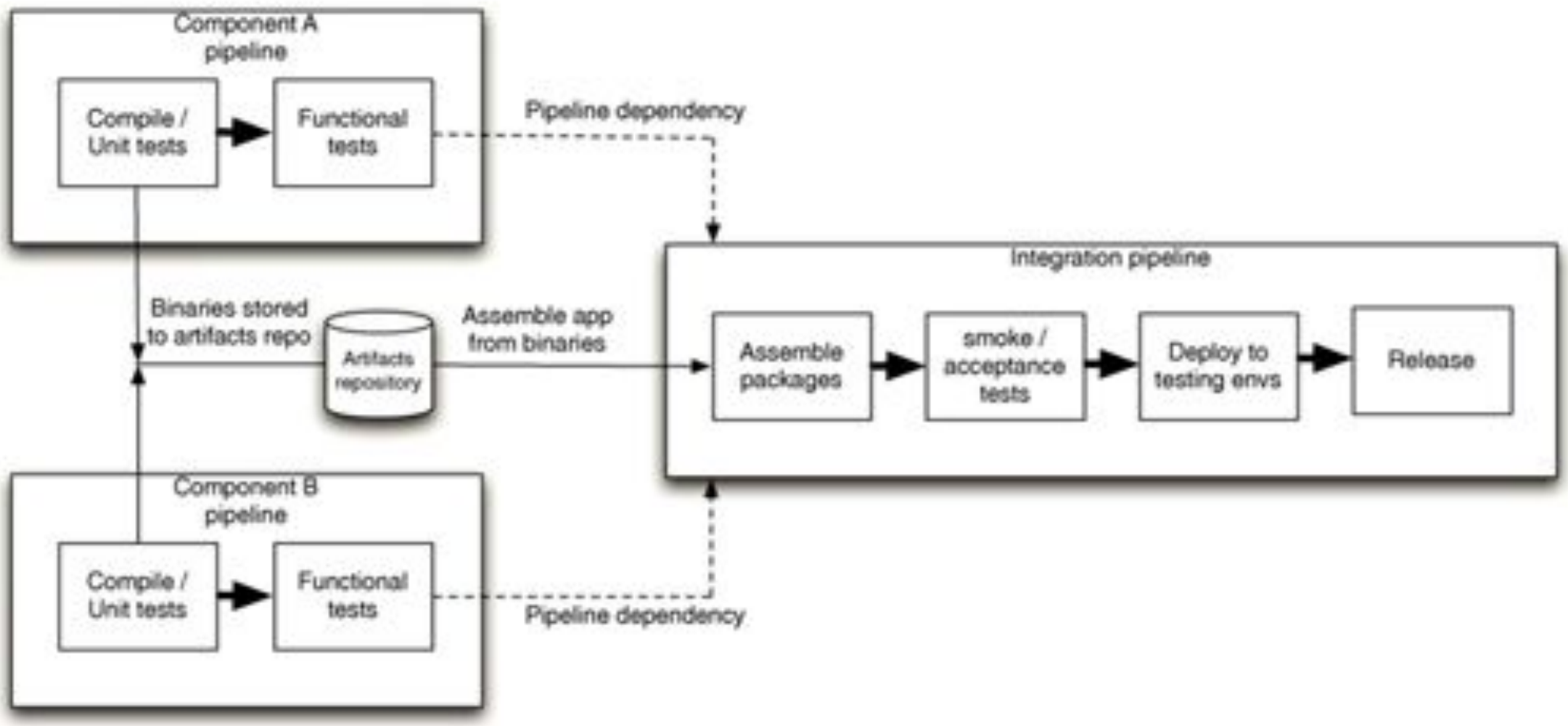
```
/**
 * Tests that a single package does not contain
 * any package dependency cycles.
 */
public void testOnePackage() {

    jdepend.analyze();

    JavaPackage p = jdepend.getPackage("com.xyz.ejb");
    assertEquals("Cycle exists: " + p.getName(),
            false, p.containsCycle());
}

/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

    assertEquals("Cycles exist",
                 false, jdepend.containsCycles());
}
```

Dependency Constraint

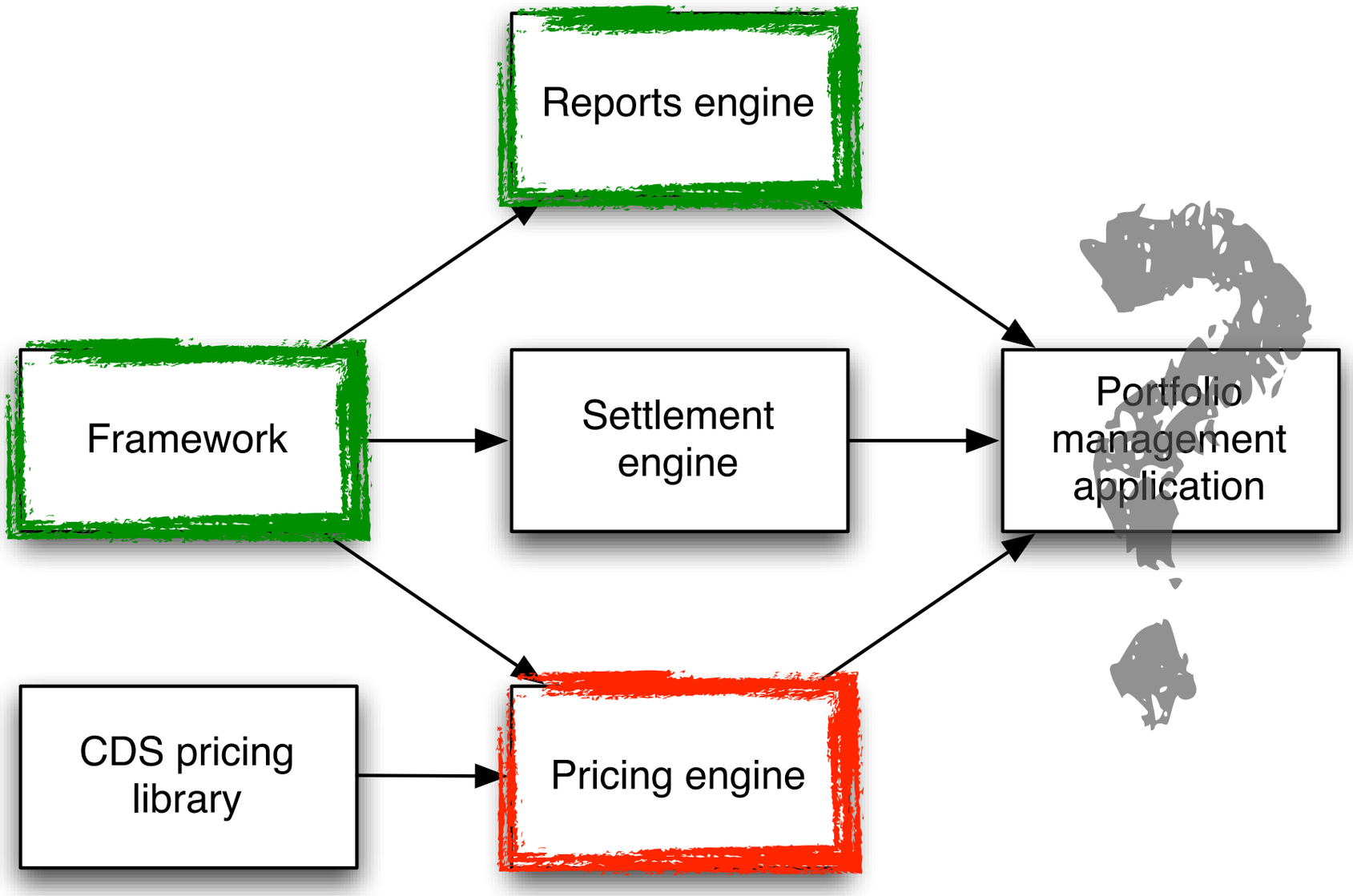
```
protected void setUp() throws IOException {
    jdepend = new JDepend();
    jdepend.addDirectory("/path/to/project/util/classes");
    jdepend.addDirectory("/path/to/project/ejb/classes");
    jdepend.addDirectory("/path/to/project/web/classes");
}
```

```
public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();
    JavaPackage ejb  = constraint.addPackage("com.xyz.ejb");
    JavaPackage web  = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    ejb.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

    assertEquals("Dependency mismatch",
            true, jdepend.dependencyMatch(constraint));
}
```

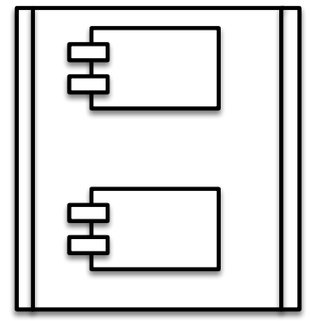
Pipelining Libraries

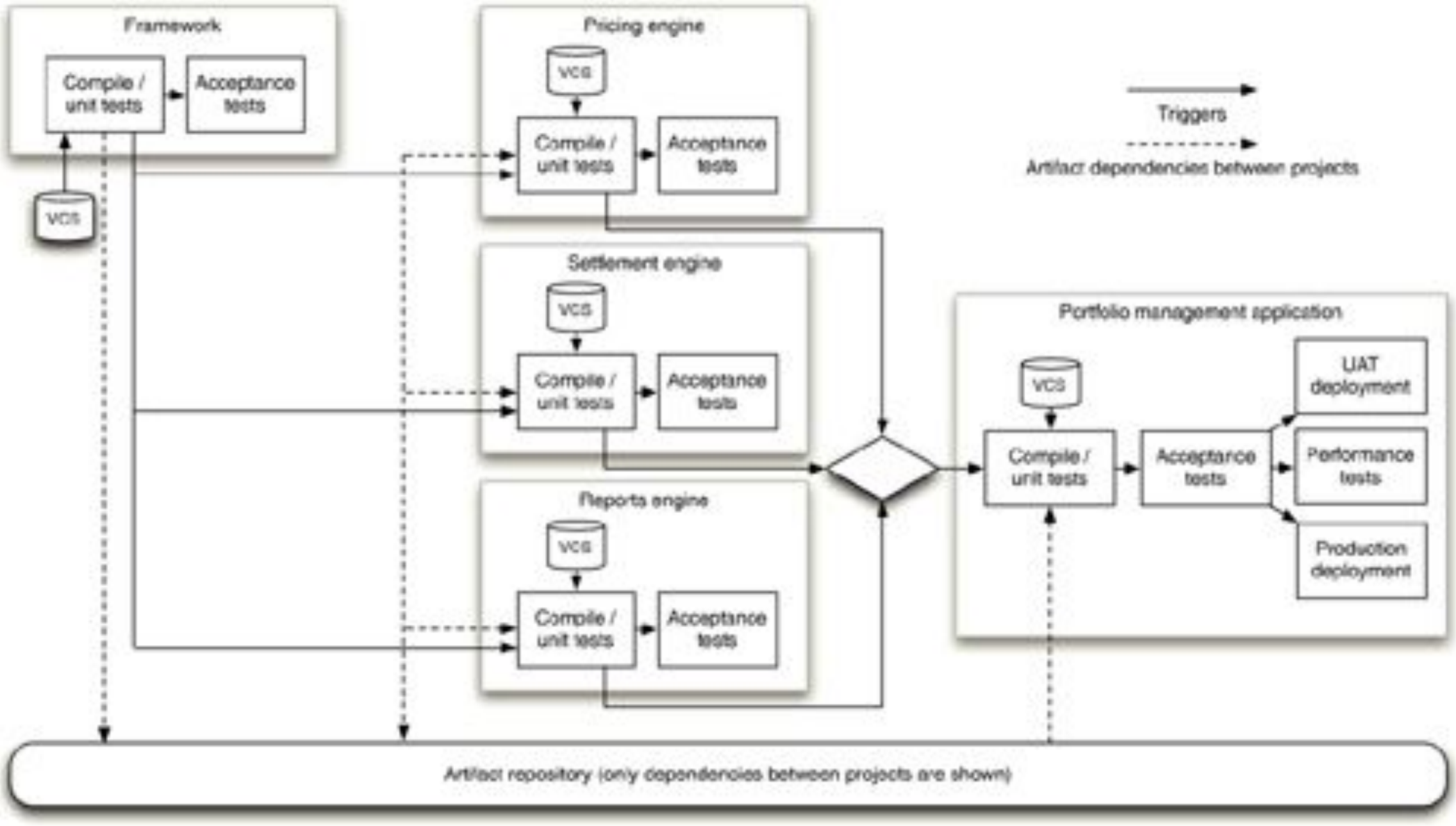


Anti-pattern: Diamond Dependencies

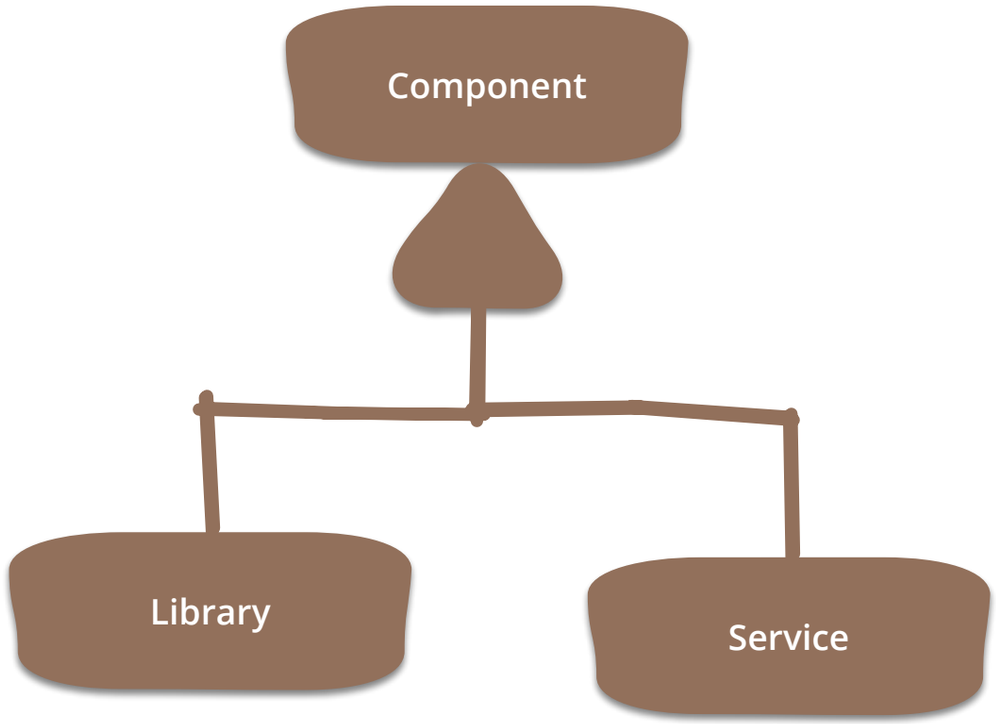


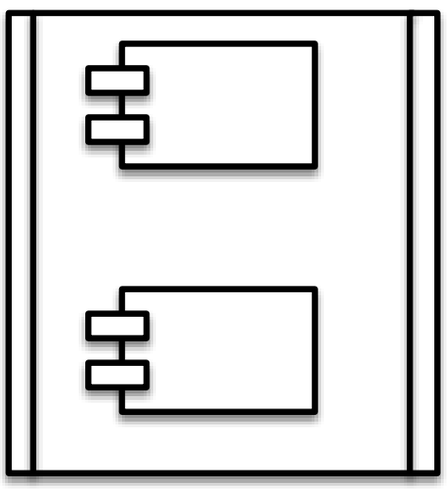
Pipelining Libraries



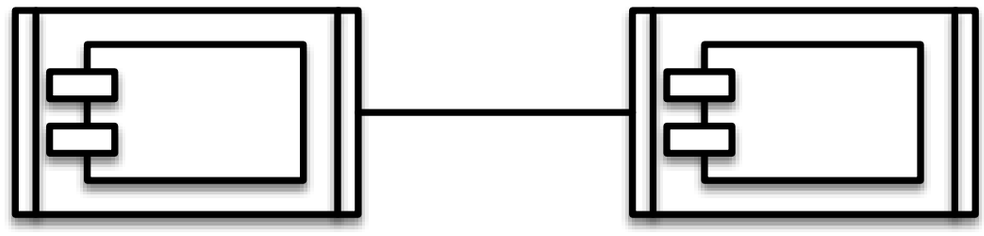


Components are units of software that can be independently replaced and upgraded



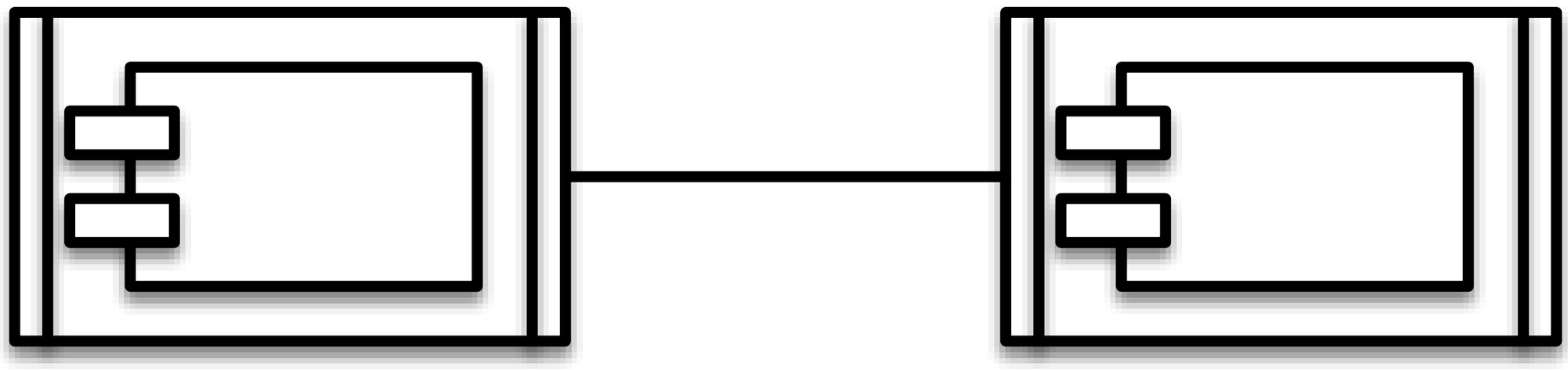


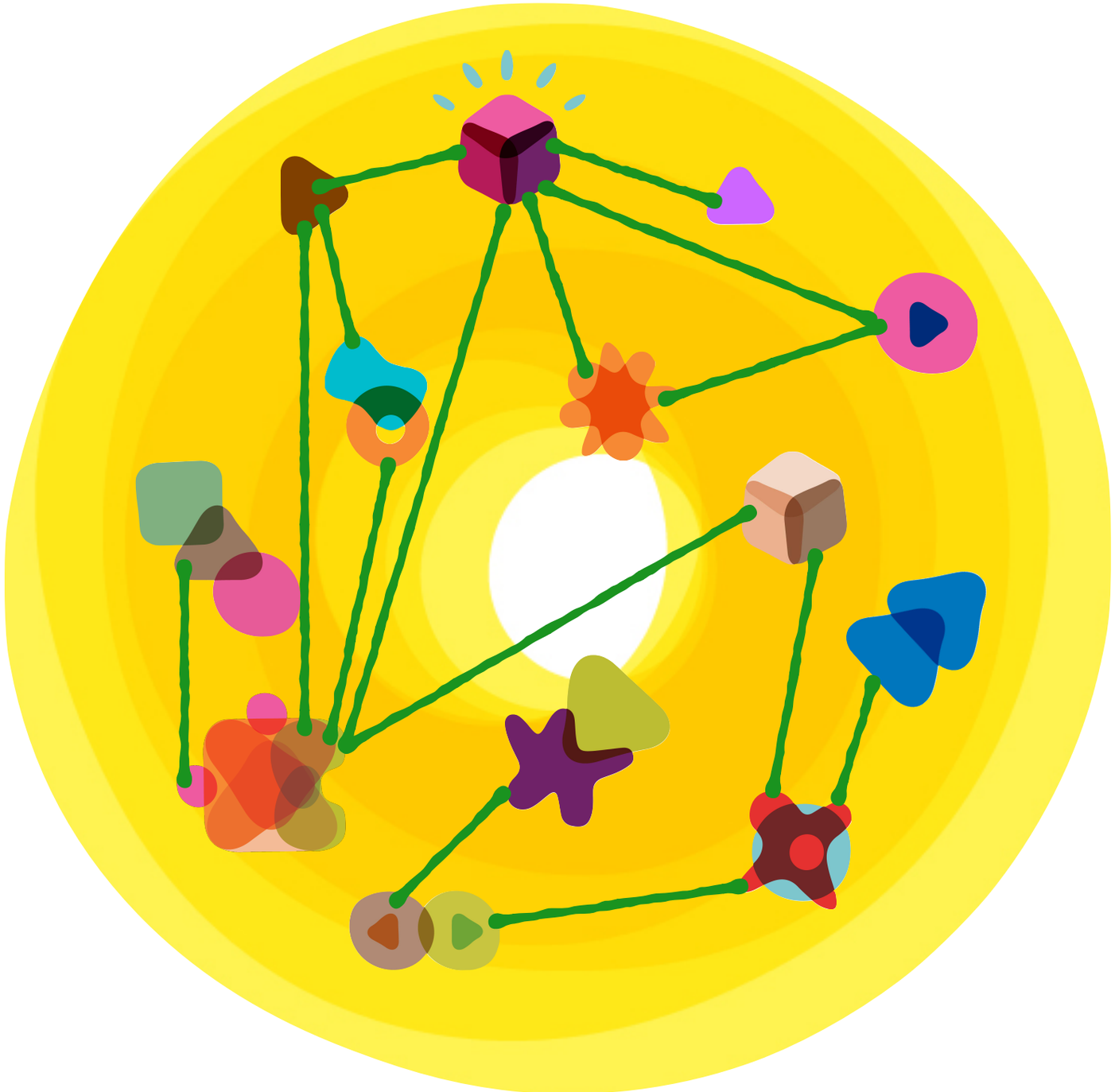
Libraries run within a single process, communicating through language function call mechanisms

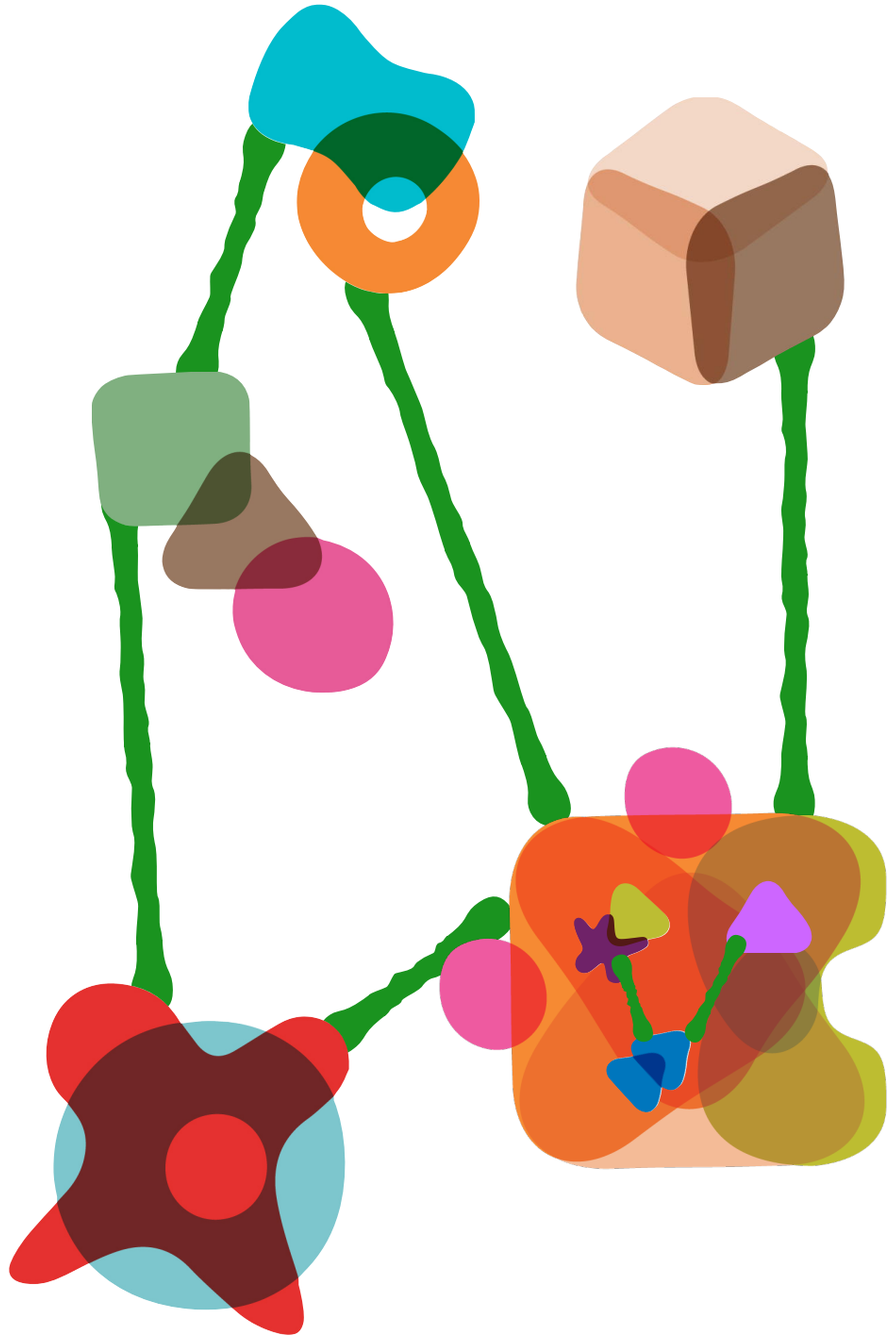


Services run in separate processes, communicating with networking mechanisms such as HTTP or TCP/IP

Services







Service-based architectures promote coupling from *application* to *integration* architecture.

en.wikipedia.org

Create account Log in

Article Talk Read Edit View history Search

Fallacies of distributed computing

From Wikipedia, the free encyclopedia

The **Fallacies of Distributed Computing** are a set of assumptions that L Peter Deutsch and others at Sun Microsystems originally asserted programmers new to distributed applications invariably make. These assumptions ultimately prove false, resulting either in the failure of the system, a substantial reduction in system scope, or in large, unplanned expenses required to redesign the system to meet its original goals.[*citation needed*]

Contents [hide]

The fallacies [edit]

The fallacies are summarized below:[1]

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator

Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

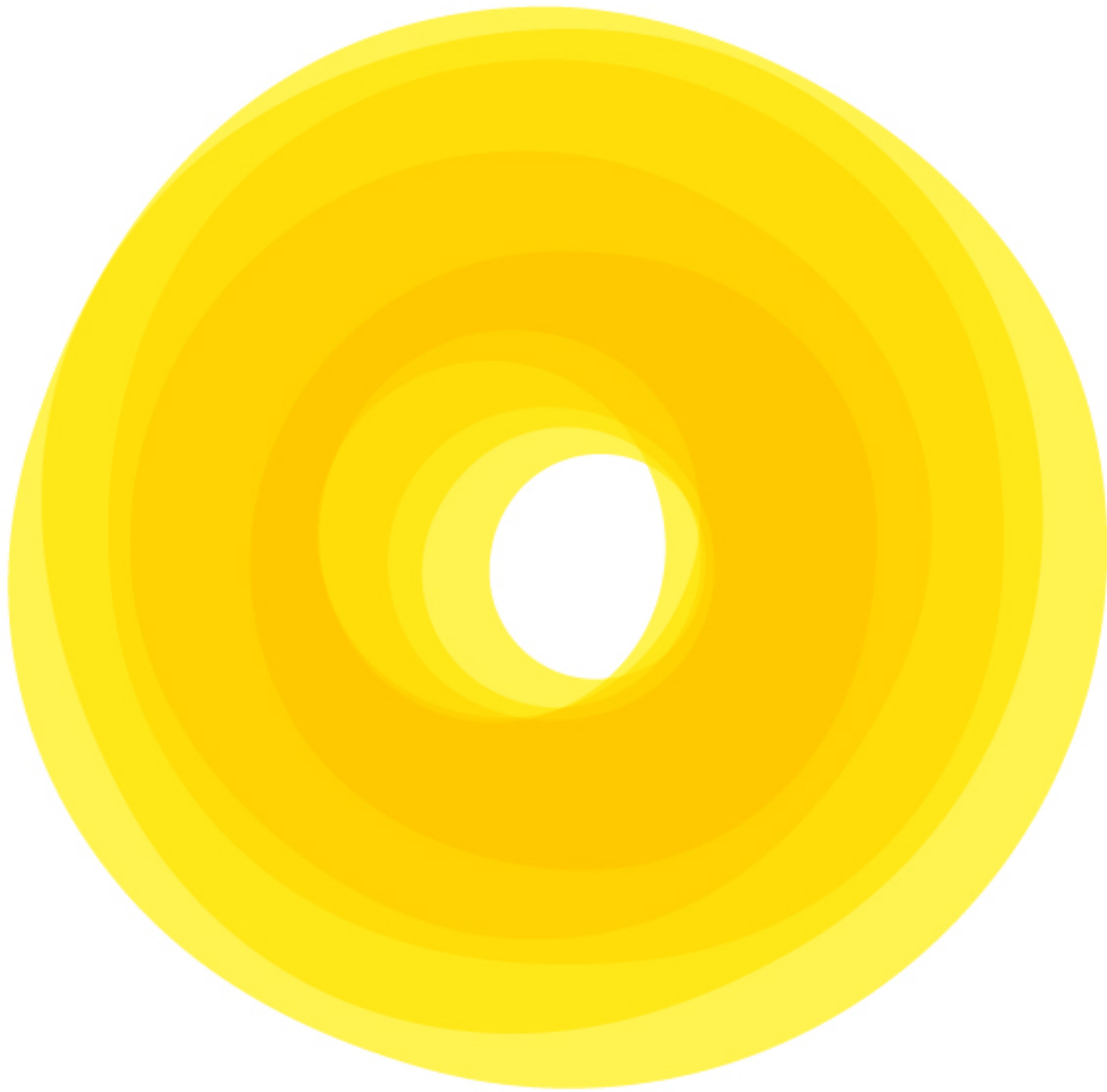
Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF

en.wikipedia.org/wiki/Fallacies_of_distributed_computing

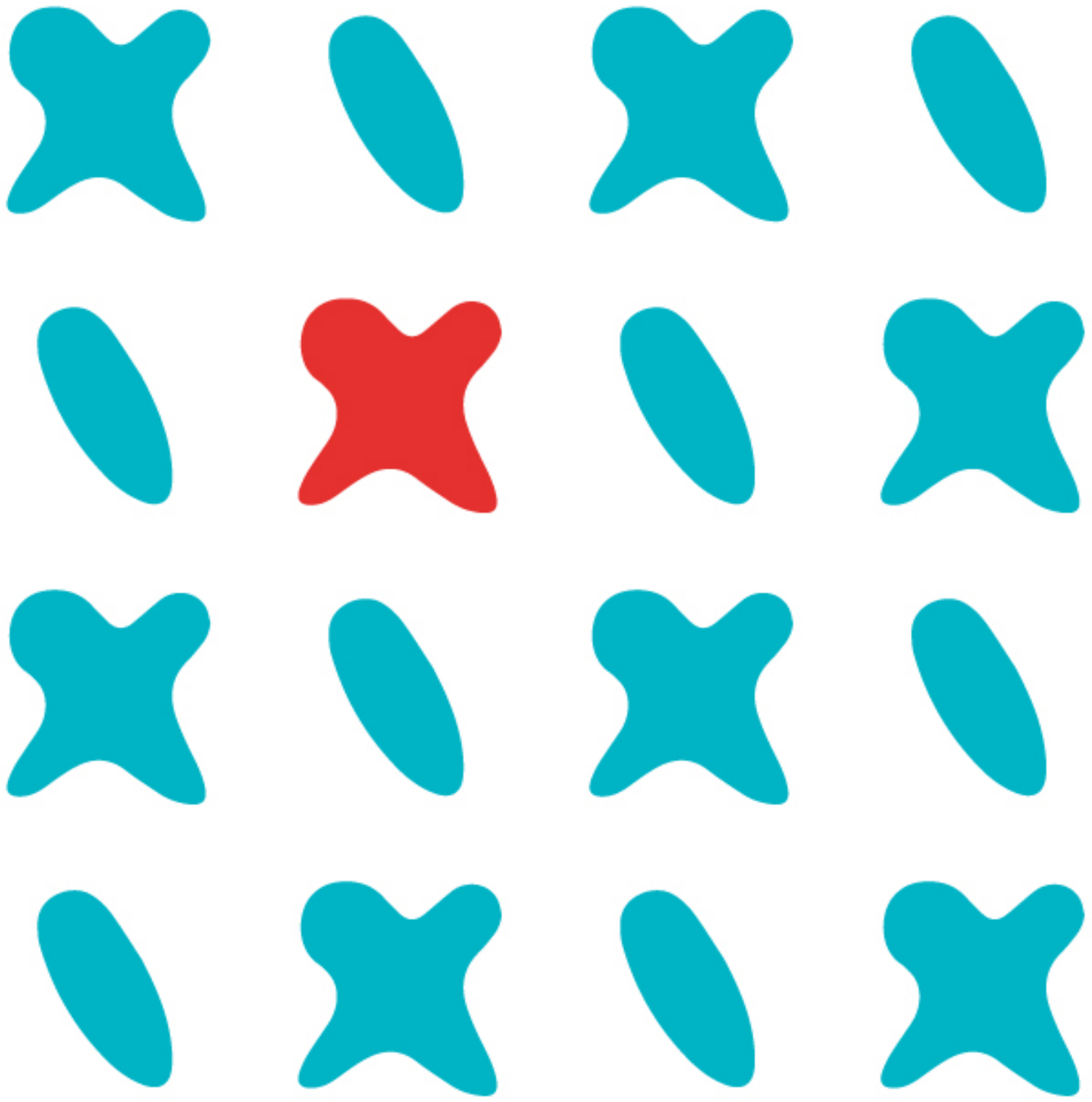
The network is reliable.

Latency is zero.



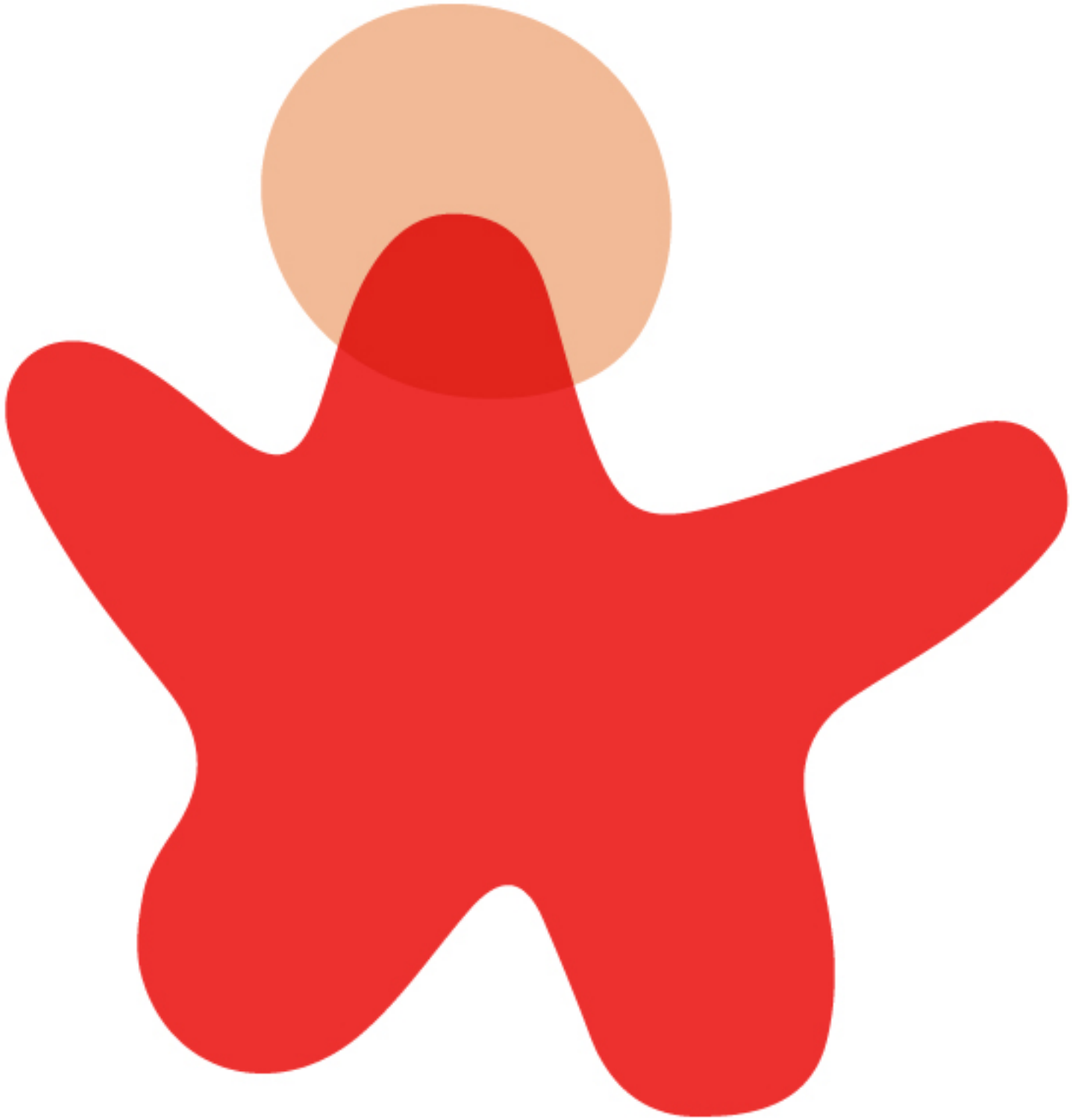
Bandwidth is infinite.

The network is secure.



Topology doesn’t change.

There is one administrator.



Transport cost is zero.

The network is homogeneous.

en.wikipedia.org

Create account Log in

Article Talk Read Edit View history Search

Fallacies of distributed computing

From Wikipedia, the free encyclopedia

The **Fallacies of Distributed Computing** are a set of assumptions that L Peter Deutsch and others at Sun Microsystems originally asserted programmers new to distributed applications invariably make. These assumptions ultimately prove false, resulting either in the failure of the system, a substantial reduction in system scope, or in large, unplanned expenses required to redesign the system to meet its original goals.[*citation needed*]

Contents [hide]

The fallacies [edit]

The fallacies are summarized below:[1]

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator

Wikipedia
The Free Encyclopedia

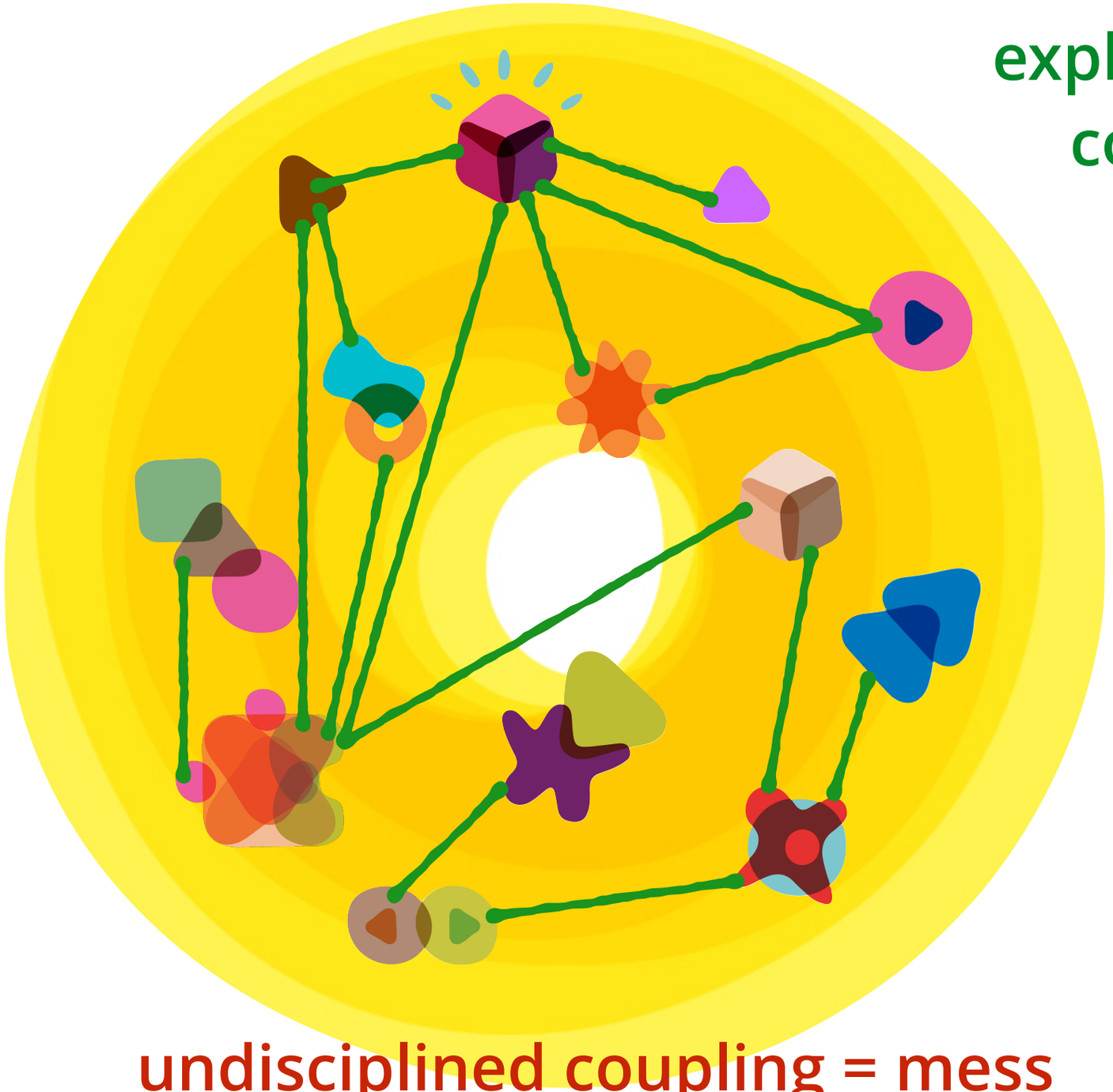
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

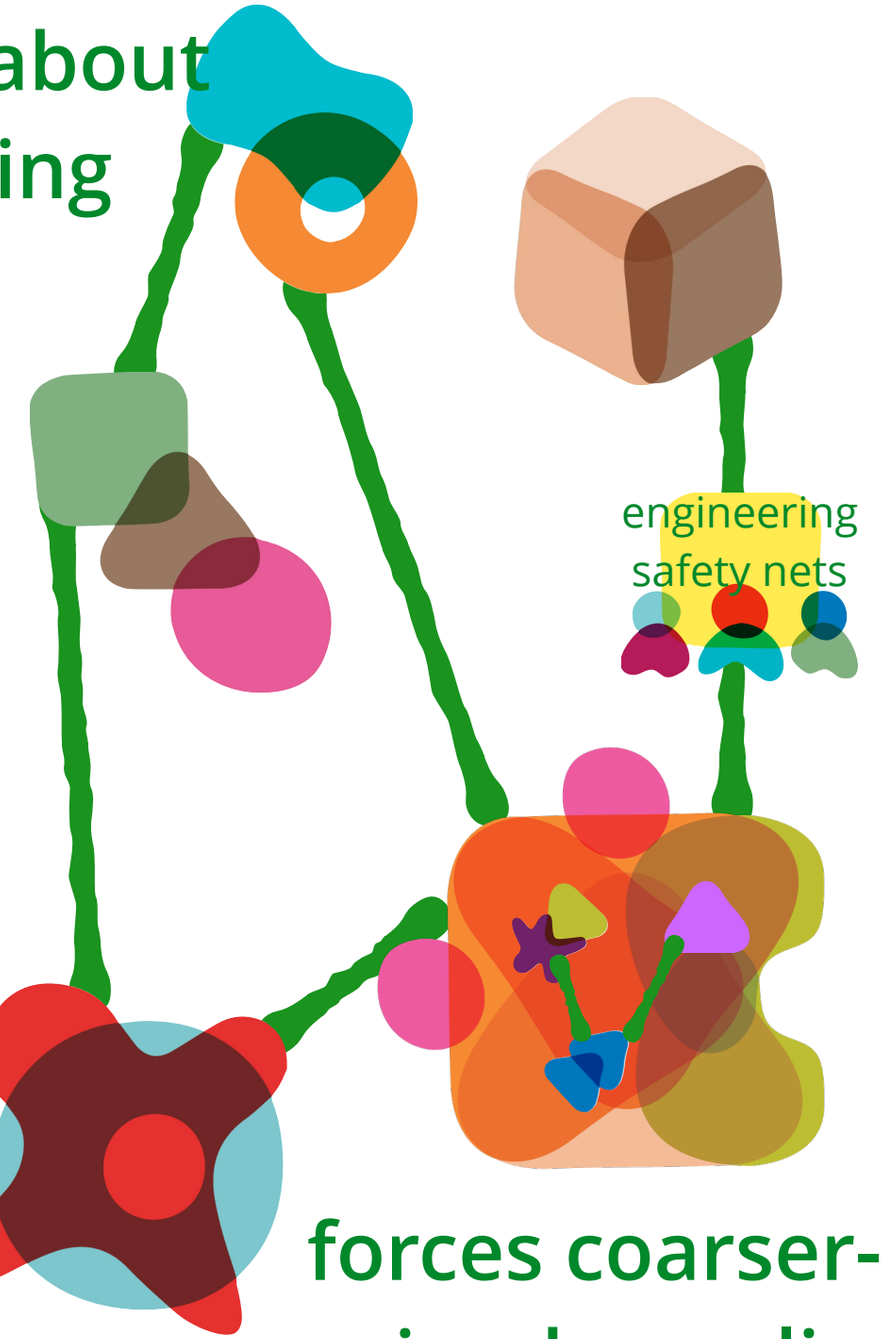
Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF

en.wikipedia.org/wiki/Fallacies_of_distributed_computing



explicit about coupling



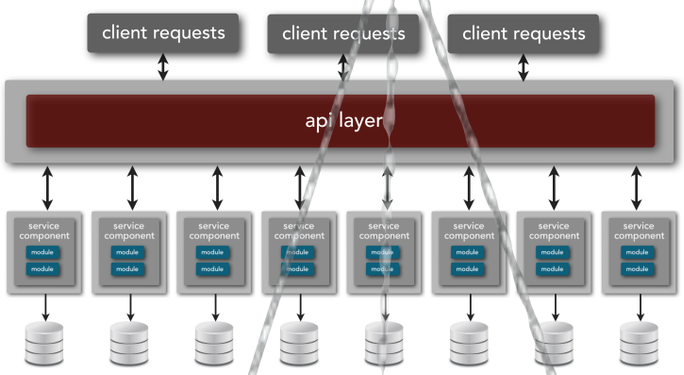
undisciplined coupling = mess

forces coarser-grained coupling

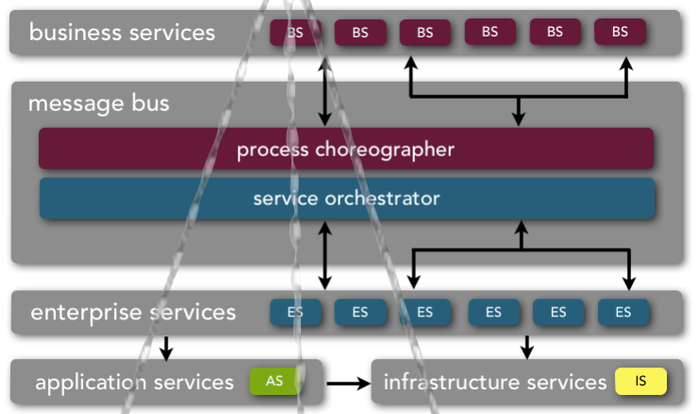
coupling dynamics become integration issues

Service-based architectures promote coupling from *application* to *integration* architecture.

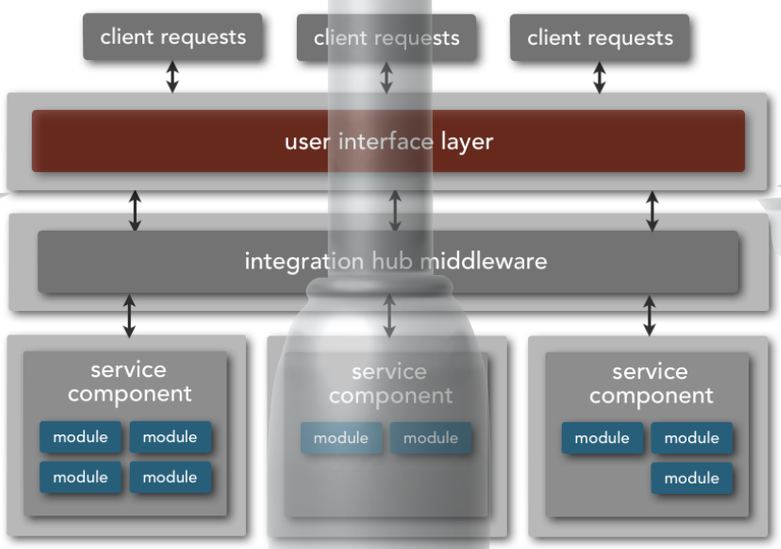
Types of SOA



Micro

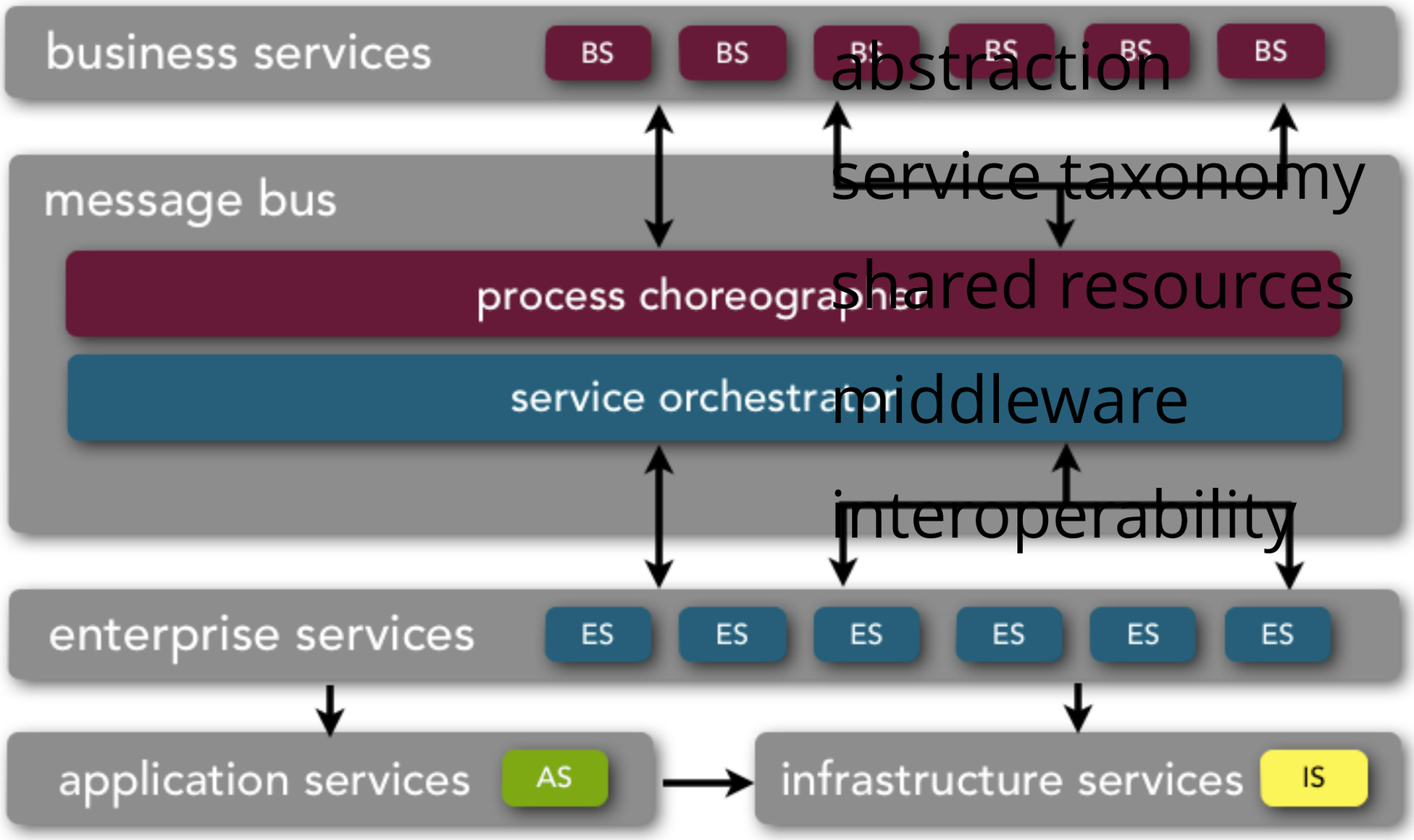


Service-oriented

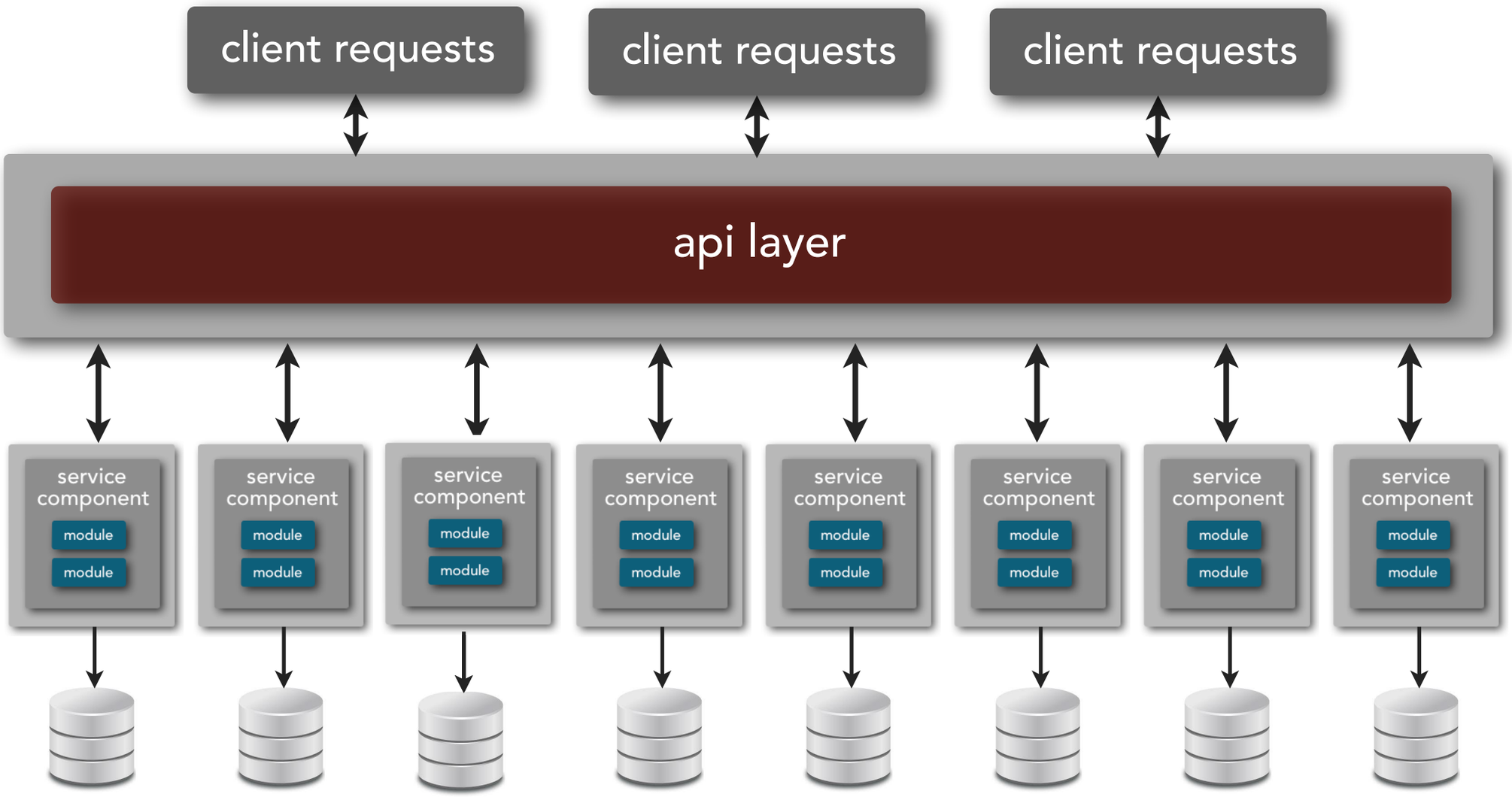


Service-based

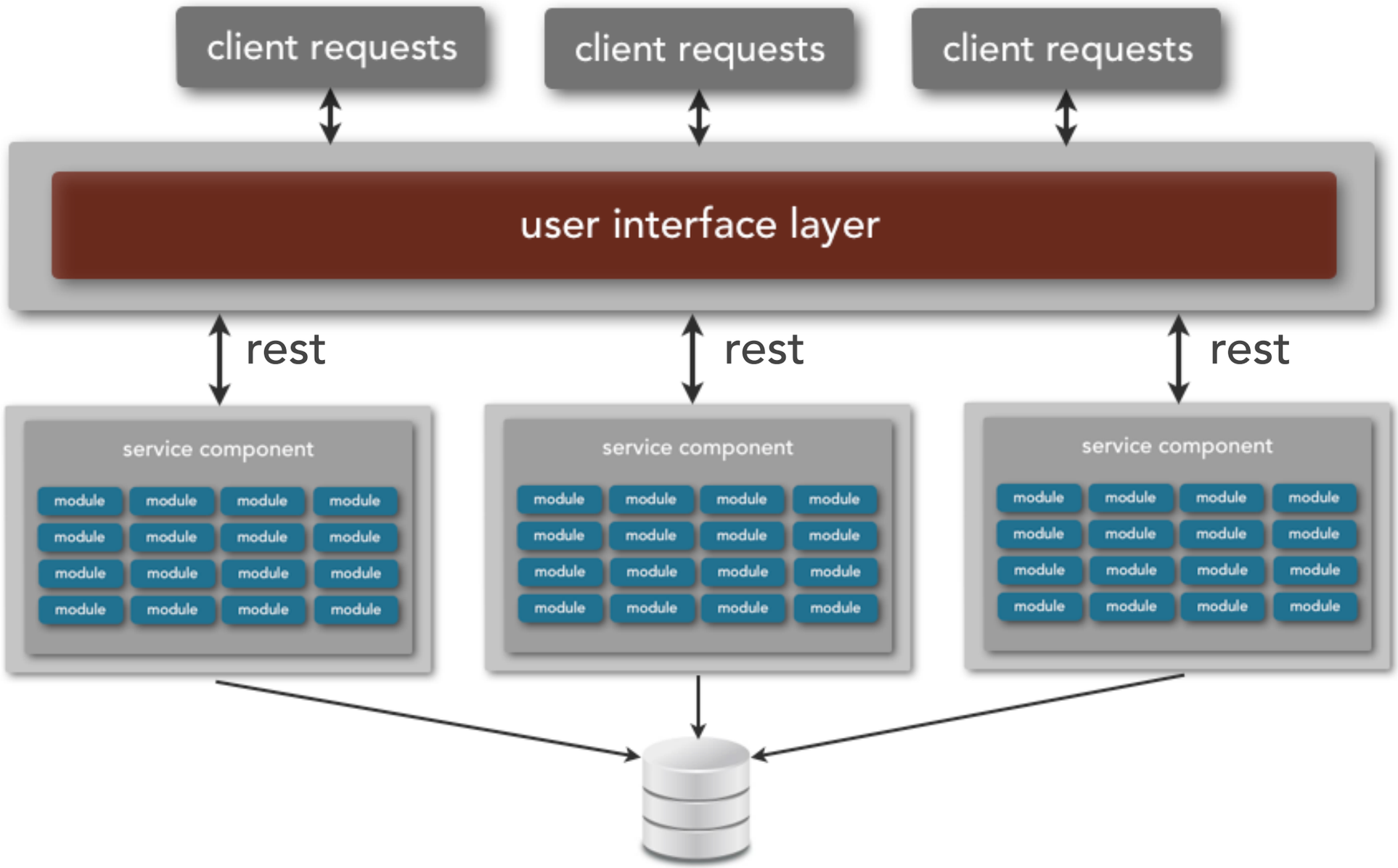
SOA (Service-oriented Architecture)



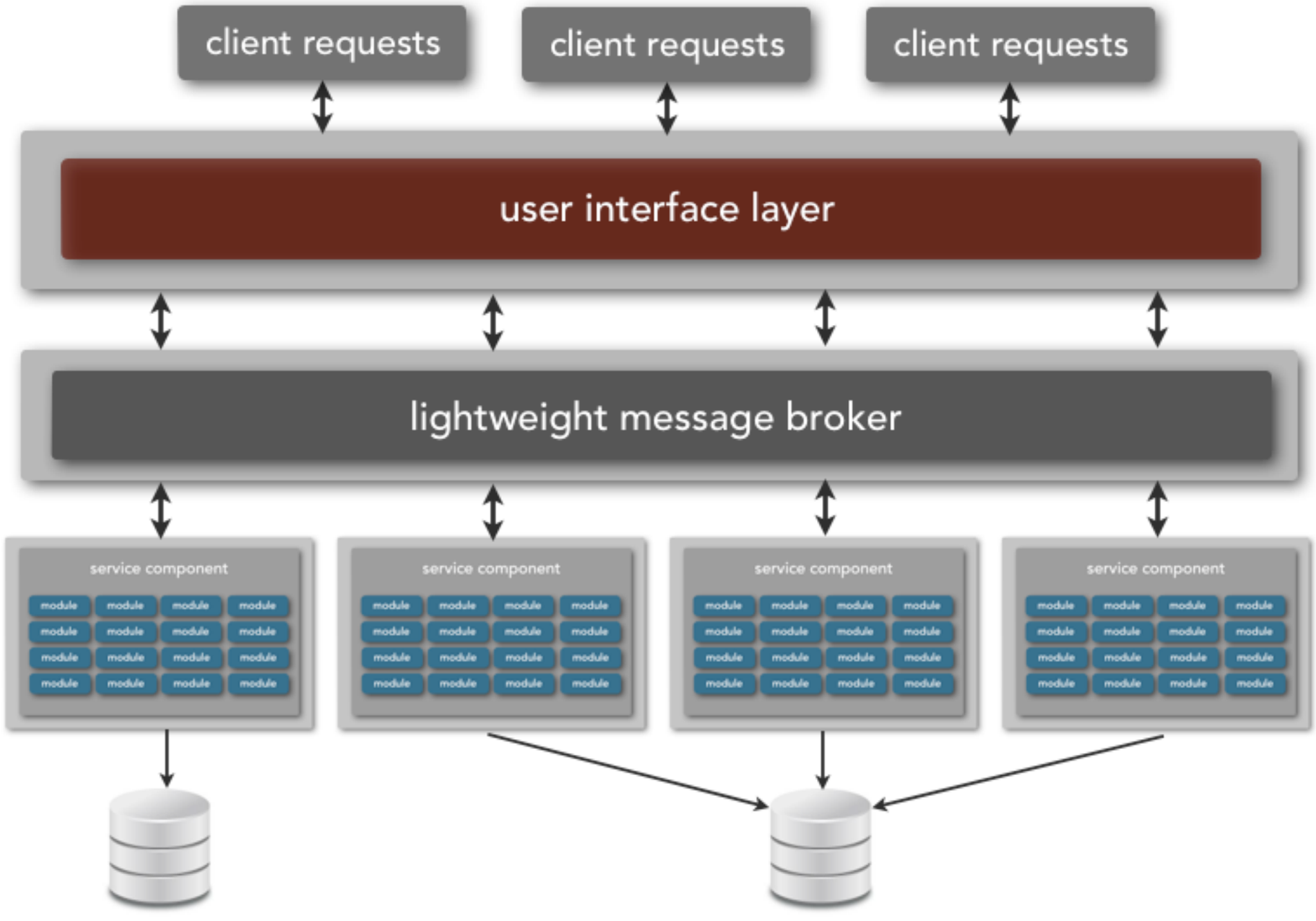
Microservice



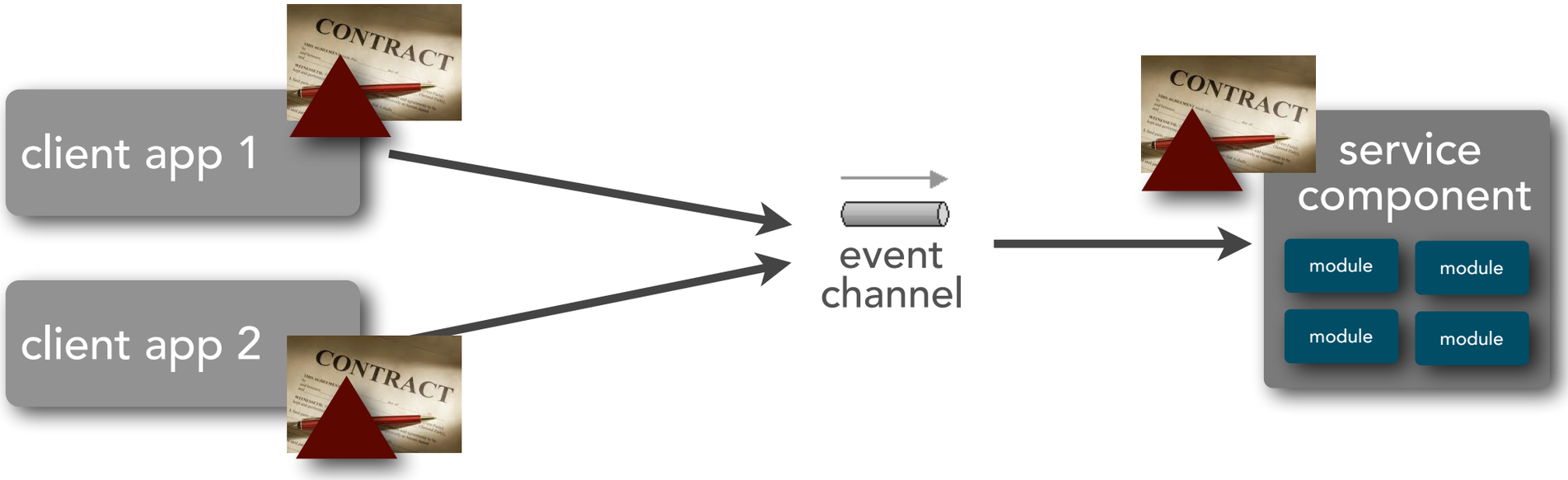
Service Based



Service Based Variants

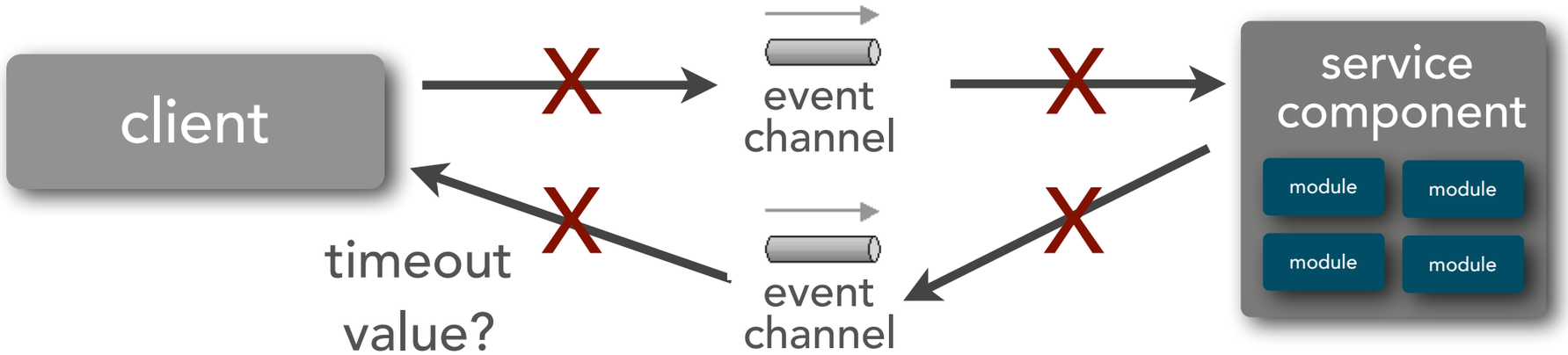


Distributed Architecture Challenges



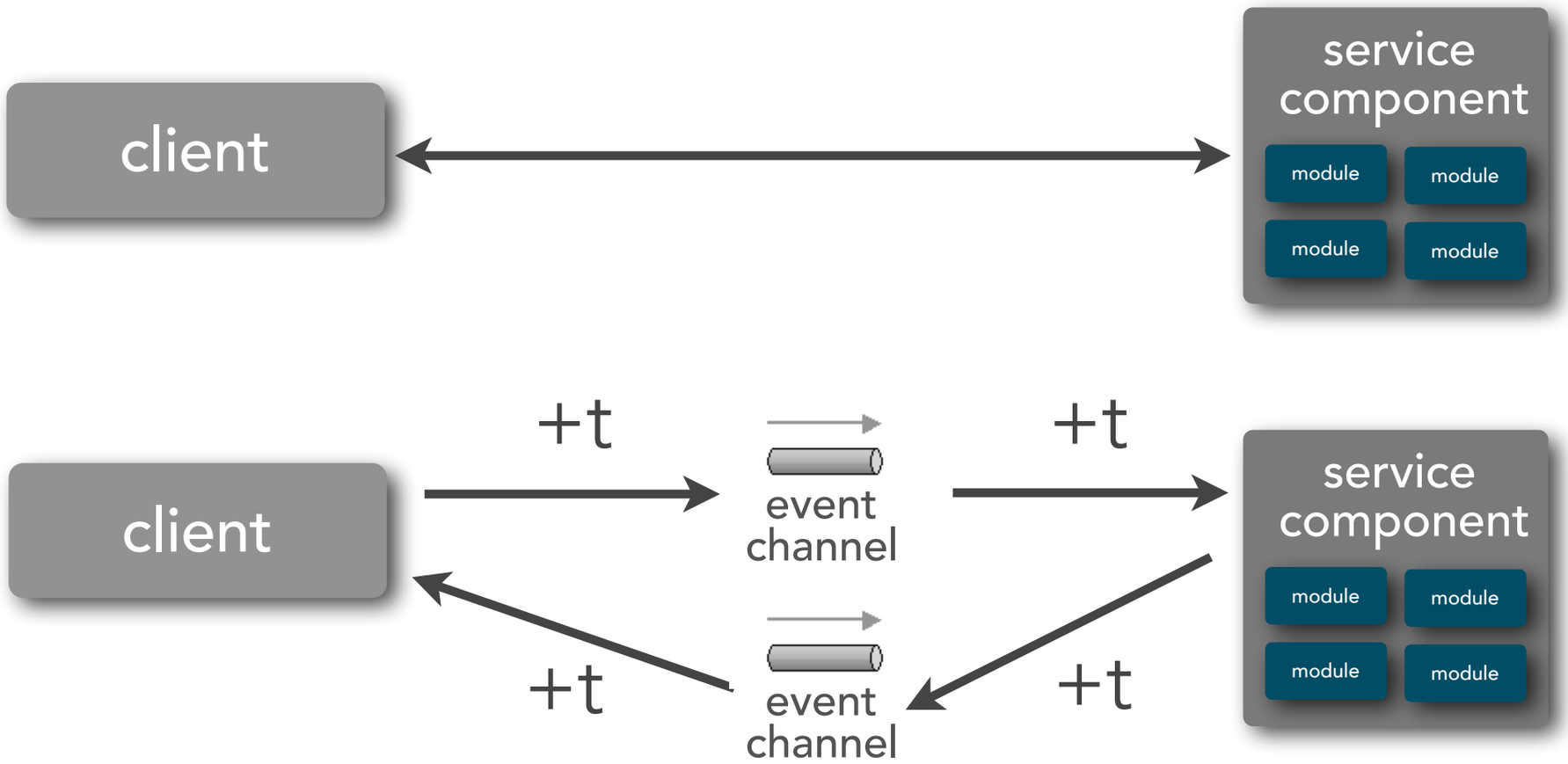
contract creation, maintenance,
versioning, and coordination

Distributed Architecture Challenges



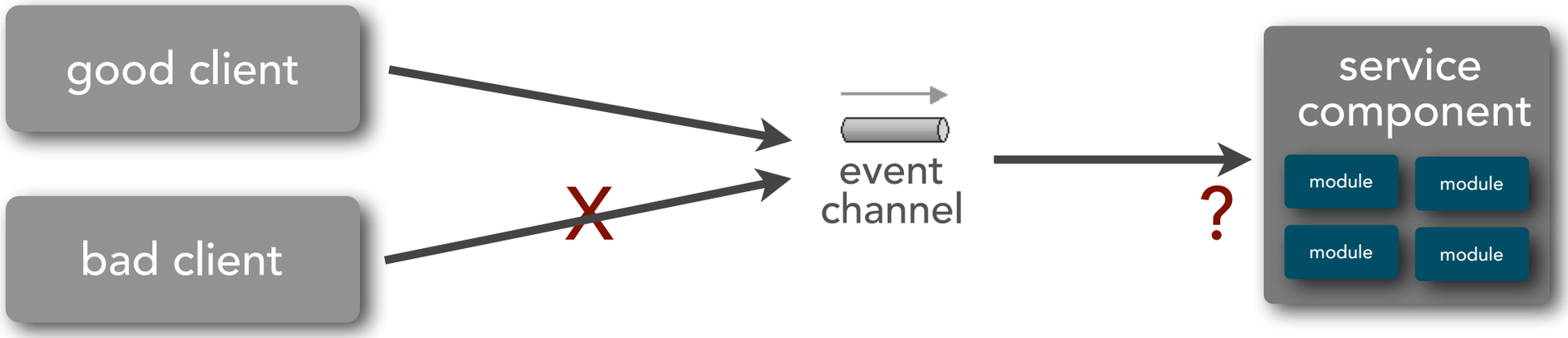
remote process responsiveness and server availability

Distributed Architecture Challenges



slower service invocations due to remote access protocols and distributed components

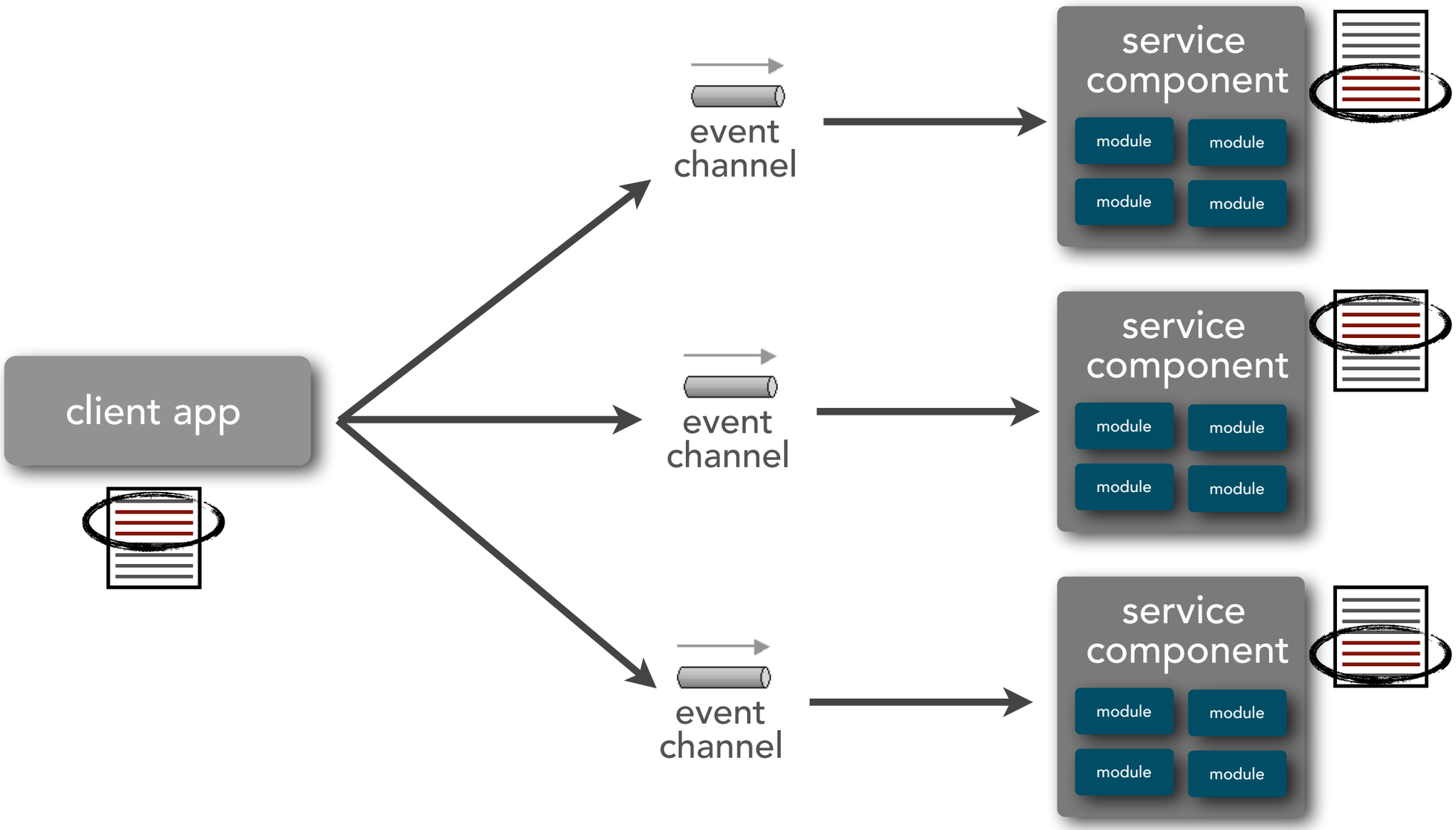
Distributed Architecture Challenges



authenticating and authorizing remote connections and service invocations

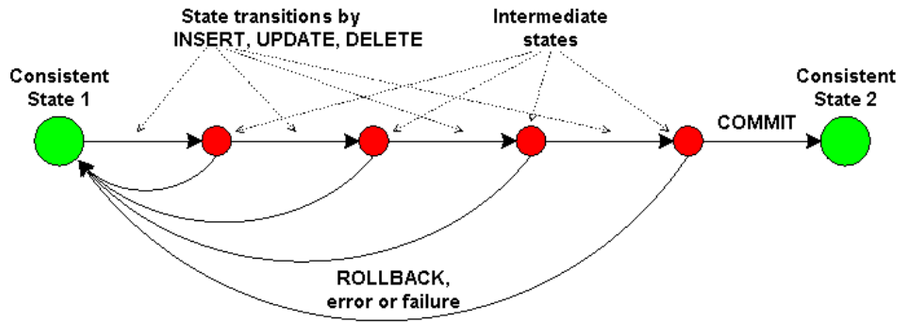
Distributed Architecture Challenges

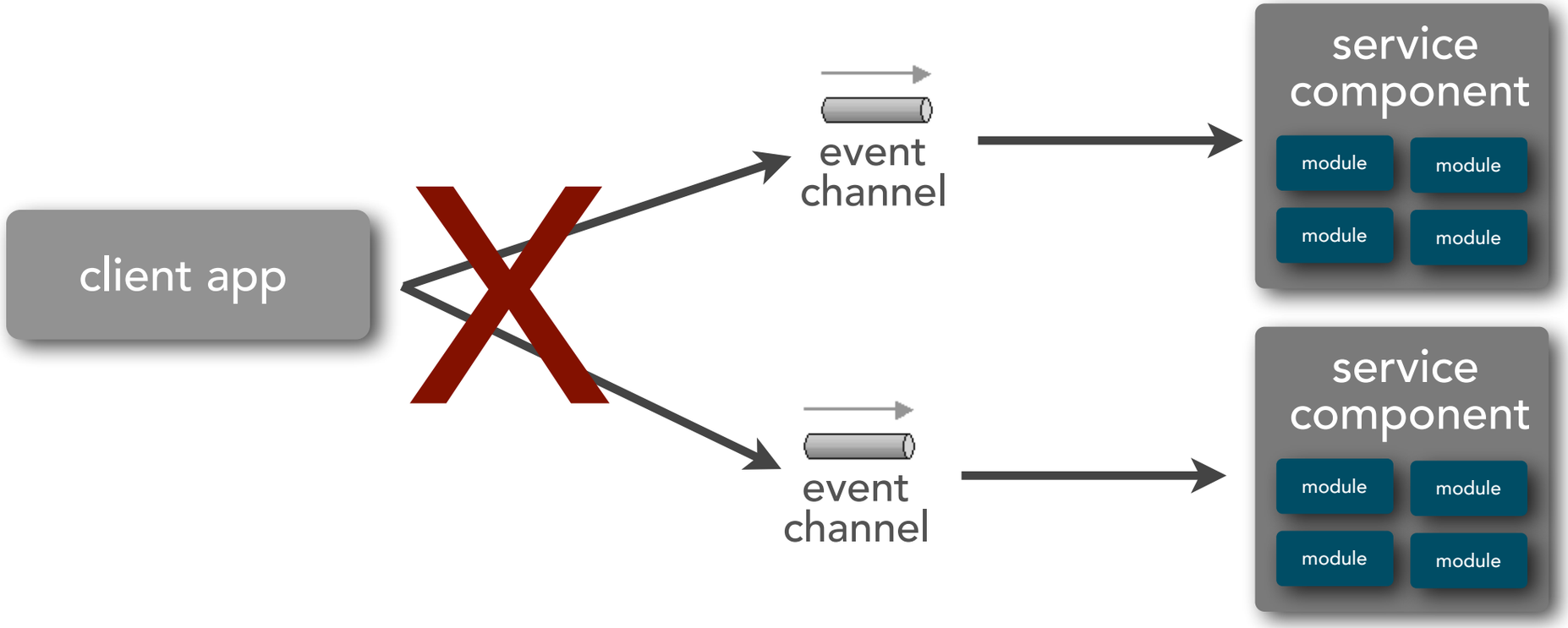
distributed logging facilities to provide a holistic view of a transaction

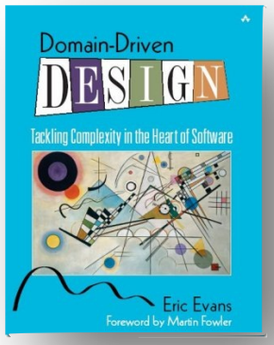


Distributed Architecture Challenges

atomic transactions and transaction scope

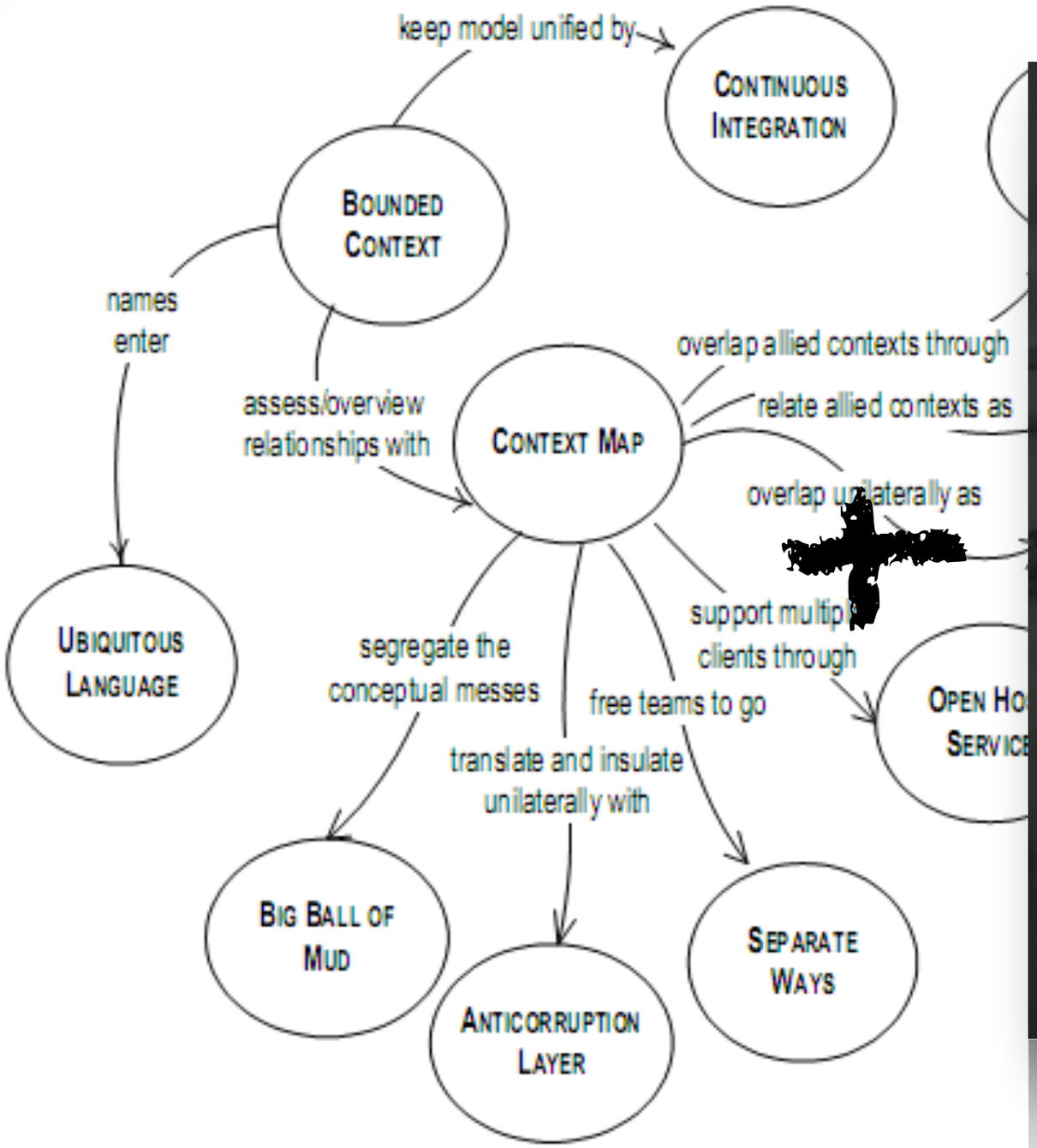


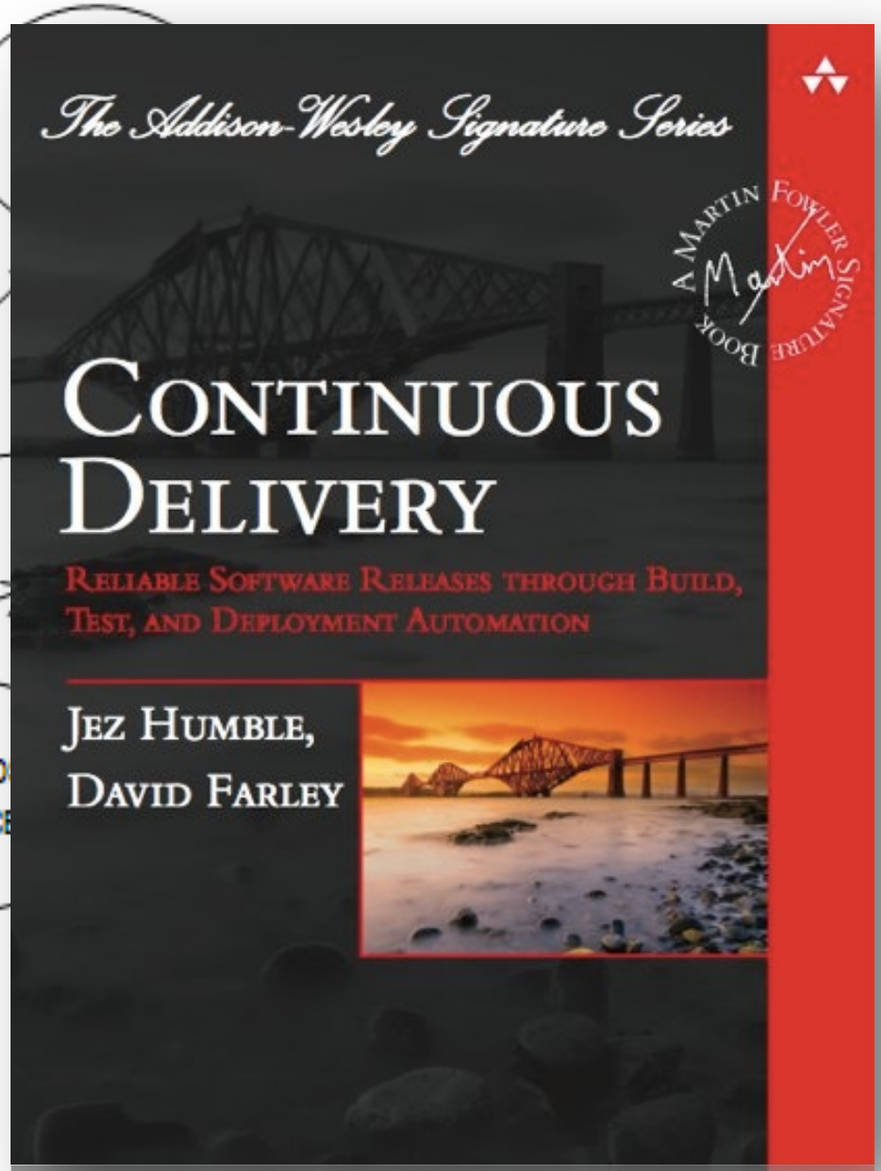




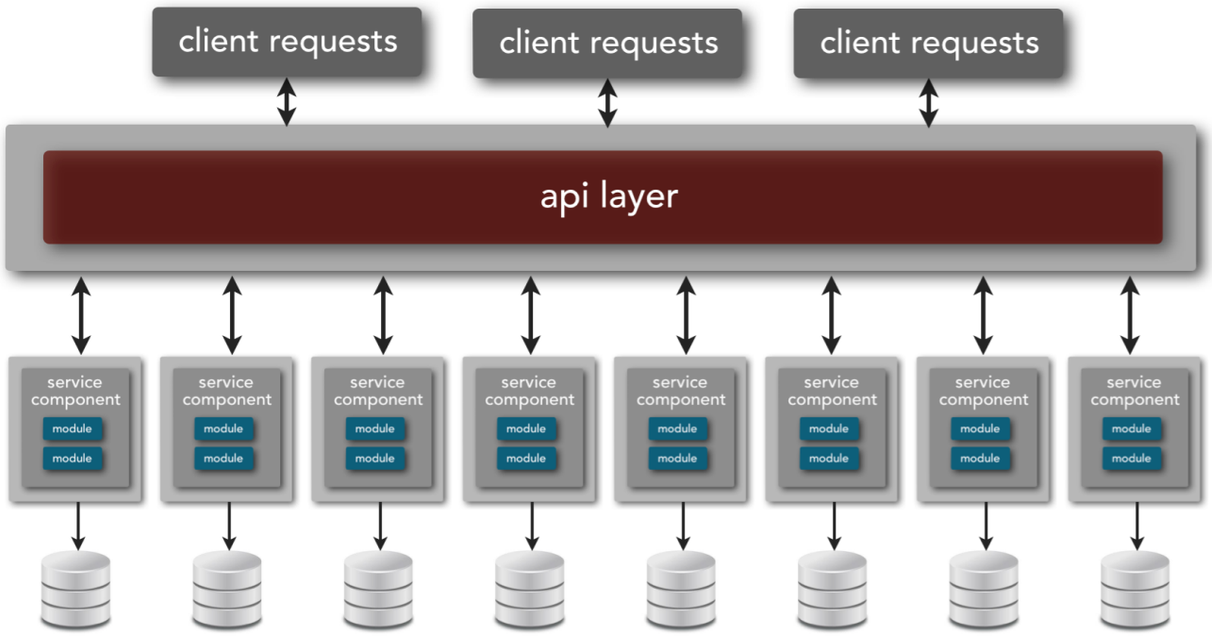
Bounded Context

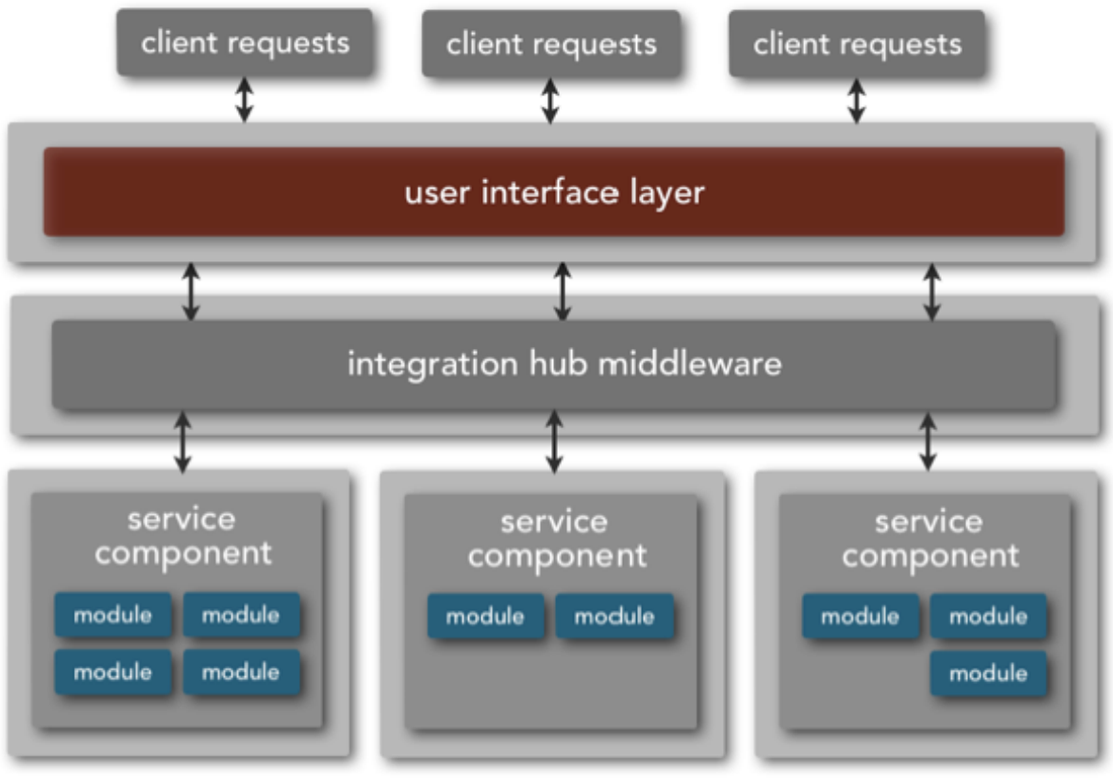
Maintaining Model Integrity





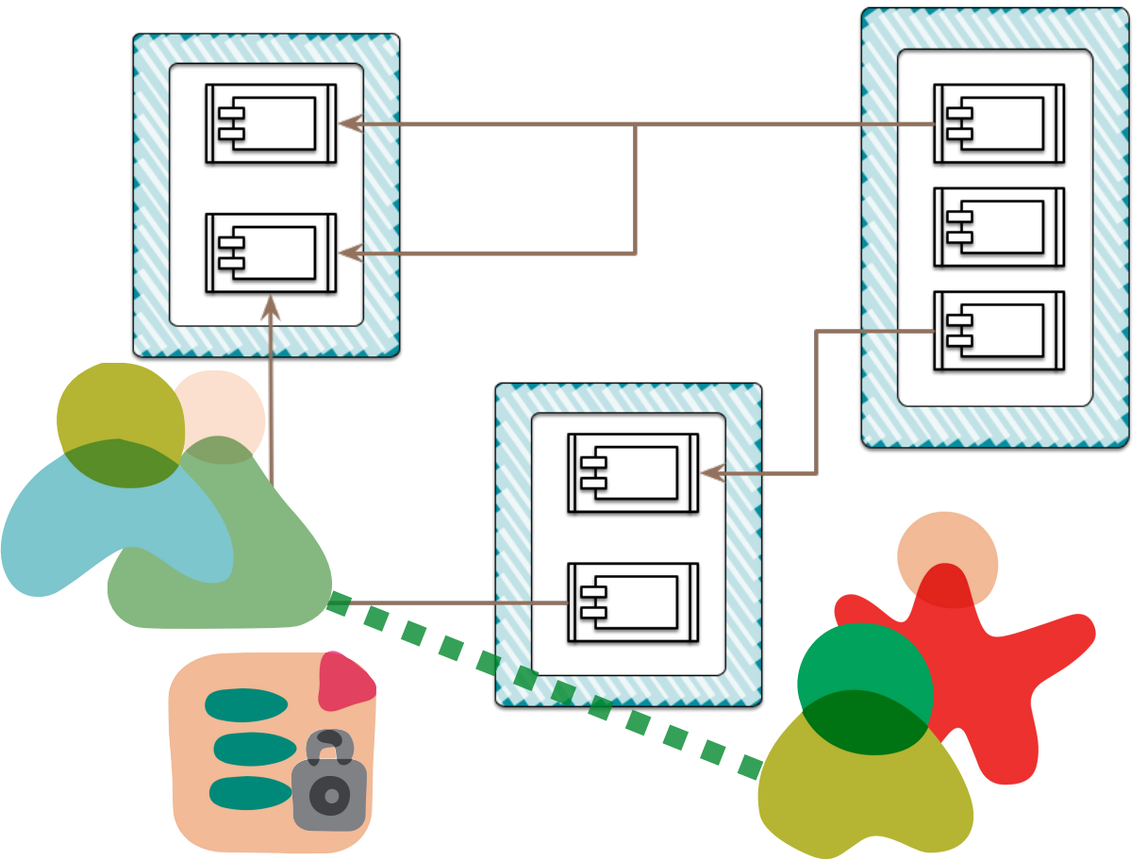
Move to Bounded Context...

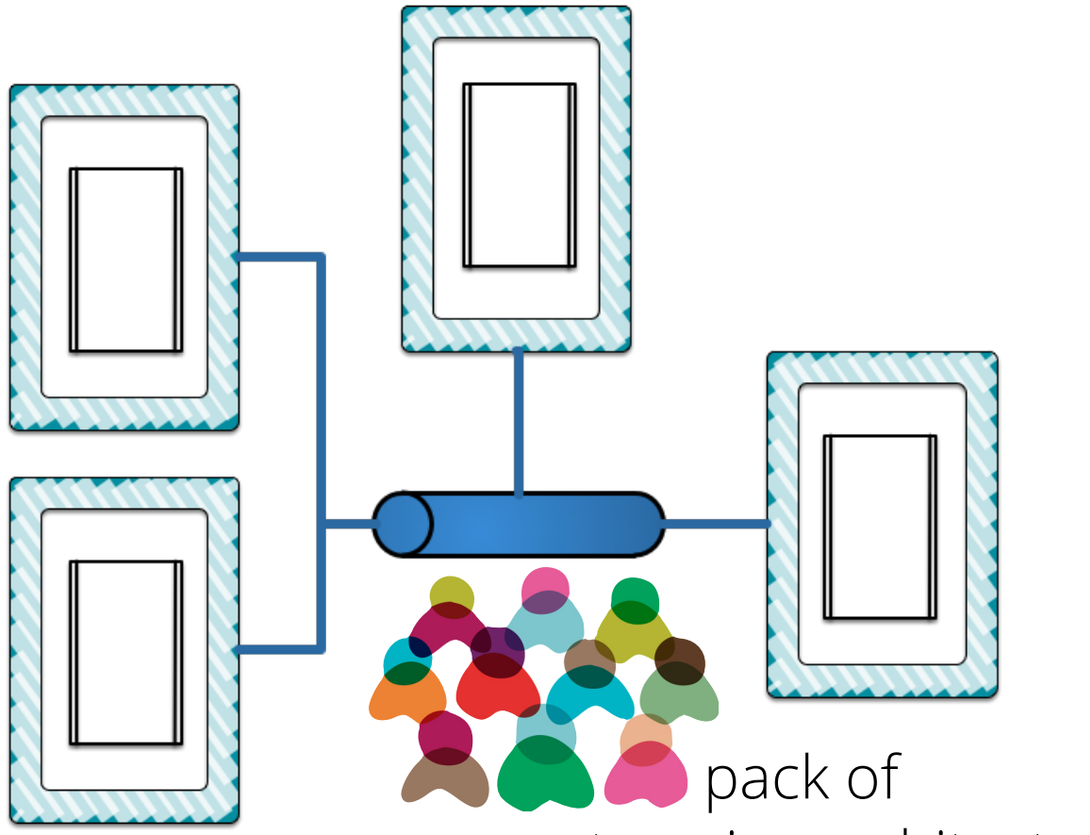




service-based

...prefer *Choreography* to *Orchestration*





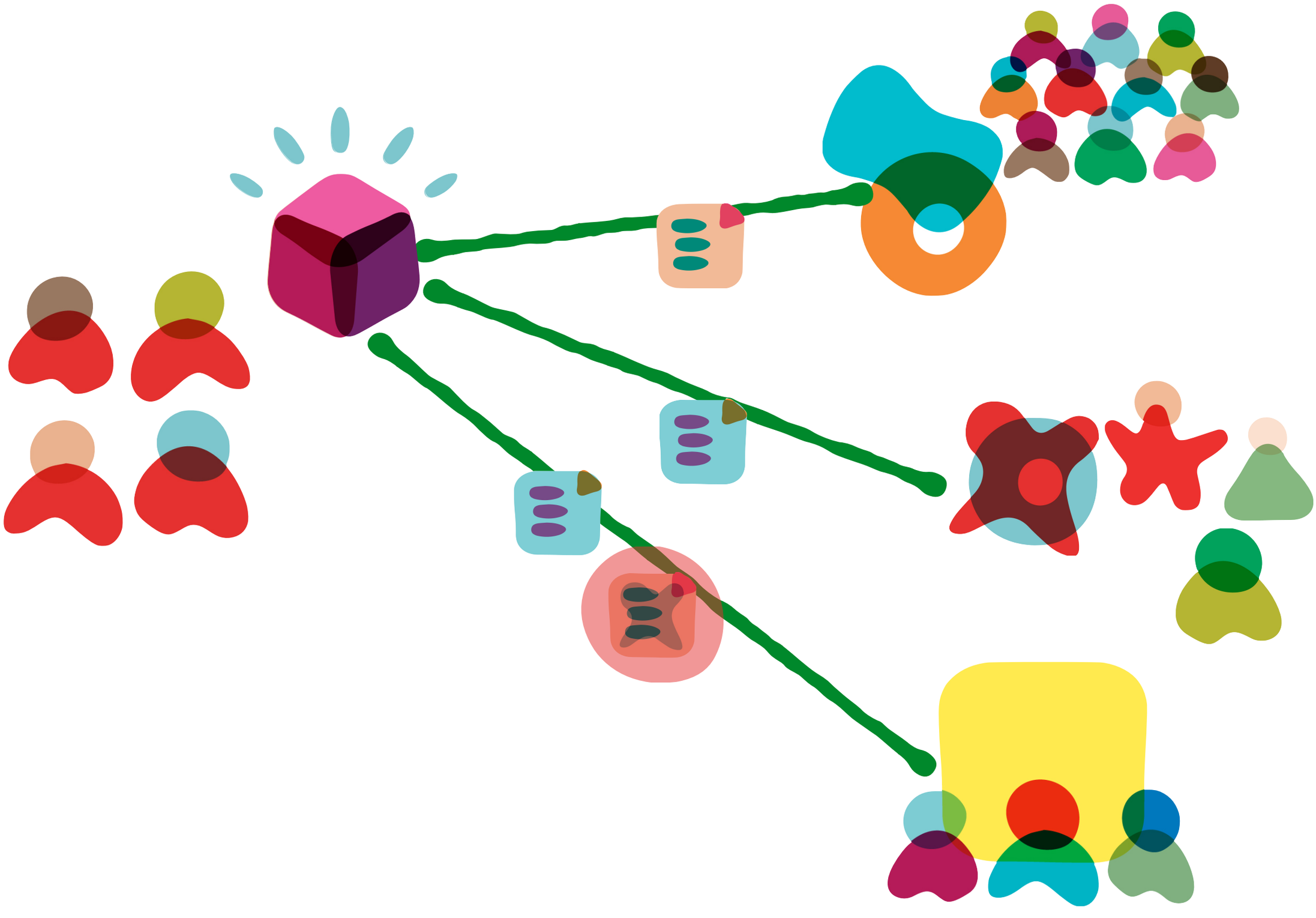
Consumer Driven Contracts

Because Conway's Law!

traditional SOA /
ESB pattern

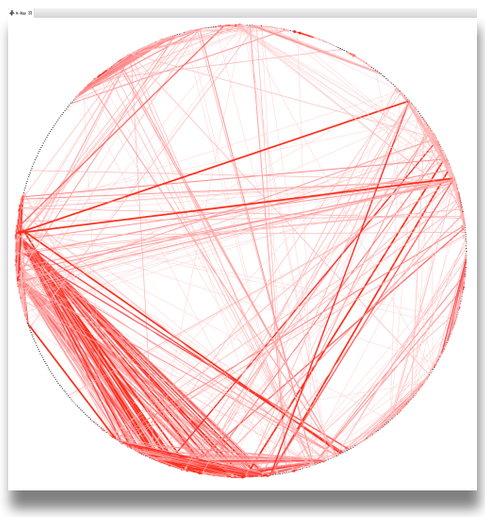
Consumer Driven Contracts

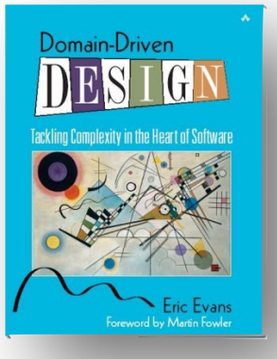
martinfowler.com/articles/consumerDrivenContracts.html



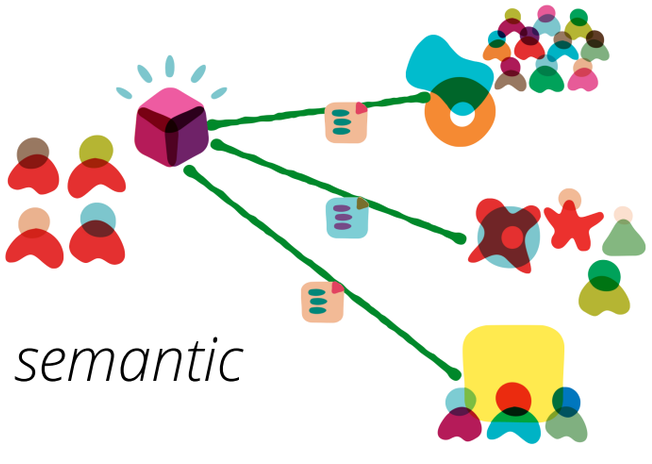
Manage coupling intelligently.

structural





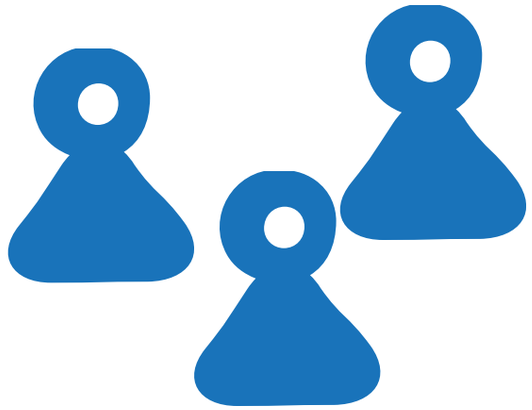
domain



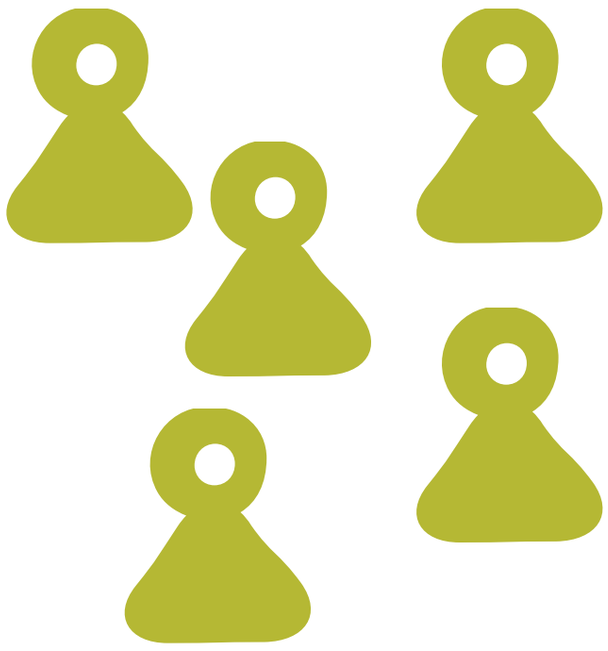
semantic

transactional

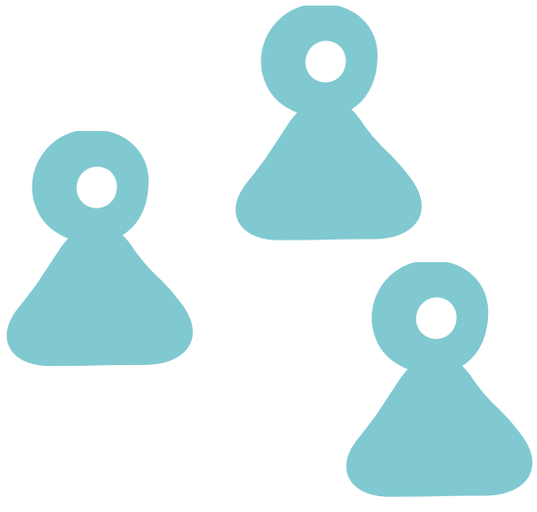
organizational



user interface



server-side

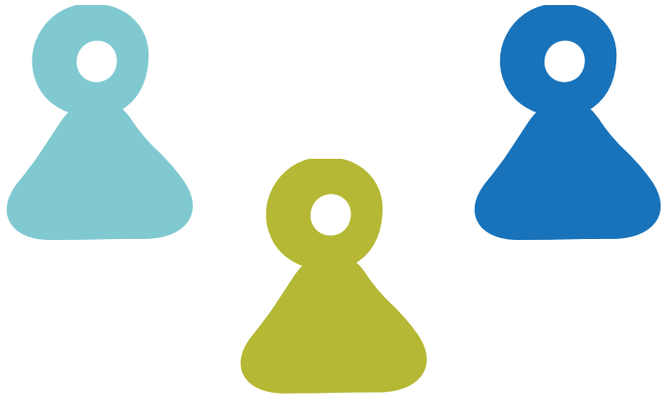


DBA

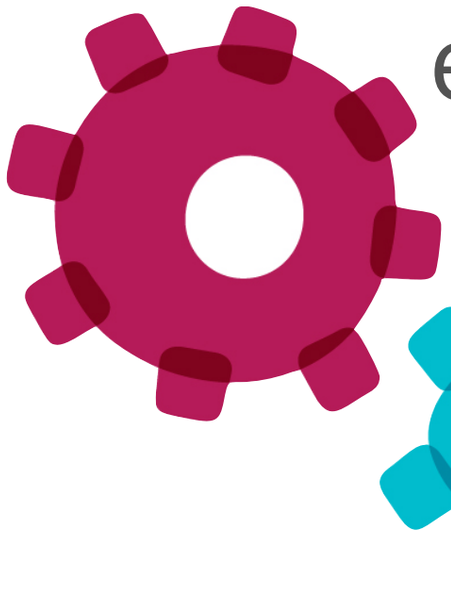
Orders

Shipping

Catalog

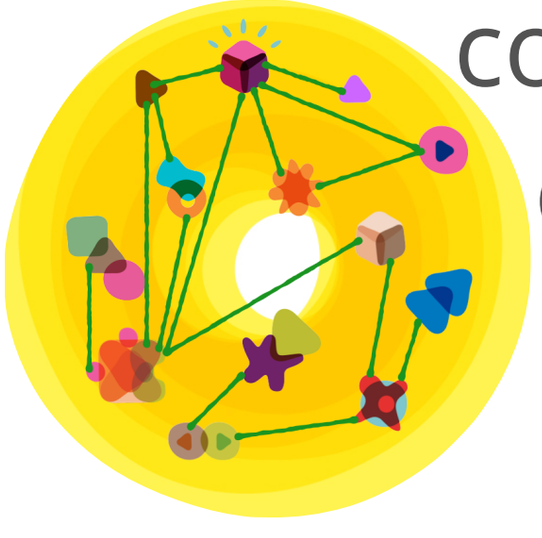


Agenda



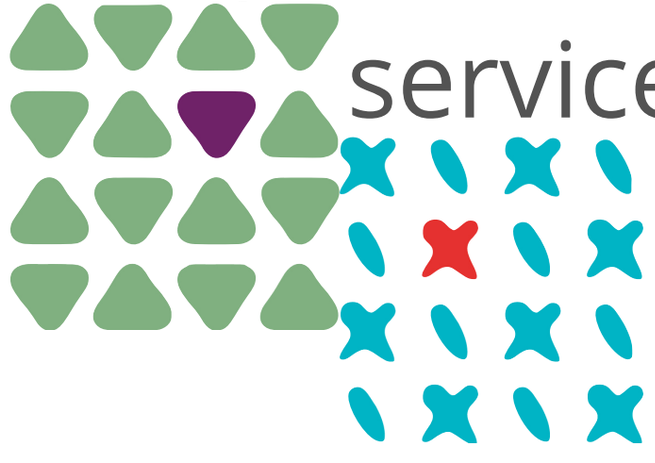
engineering practices

deployment models



coupling & cohesion

components & services



Machine Provisioning

Chef

puppet labs

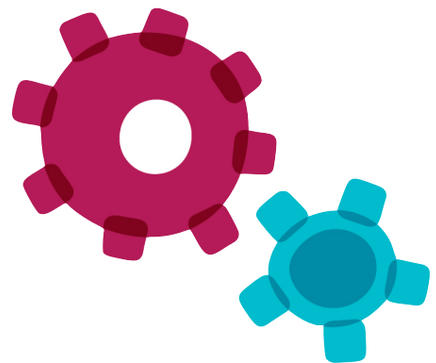
ANSIBLE

manage many systems

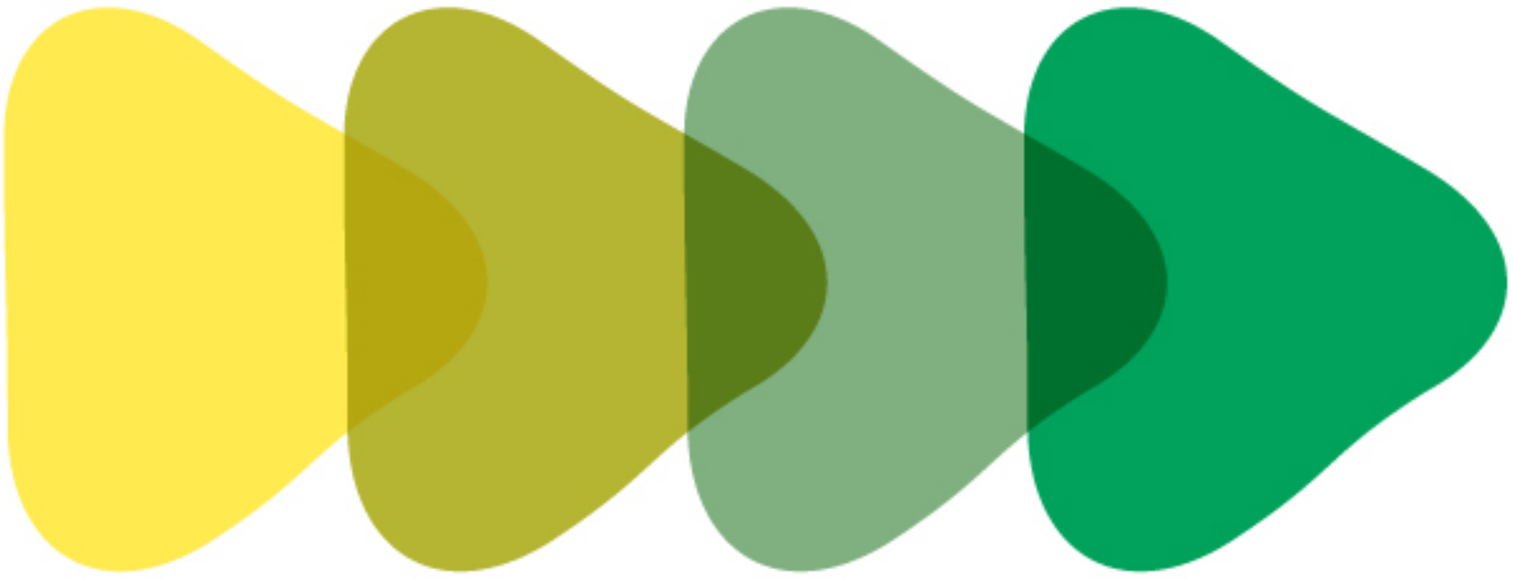
manage configuration

enforce consistency

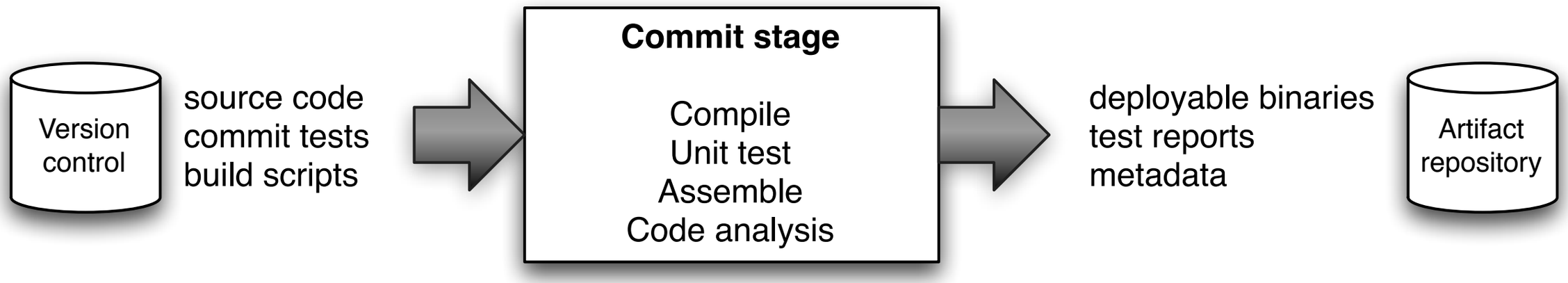
treat infrastructure as code



Deployment Pipelines



commit Stage

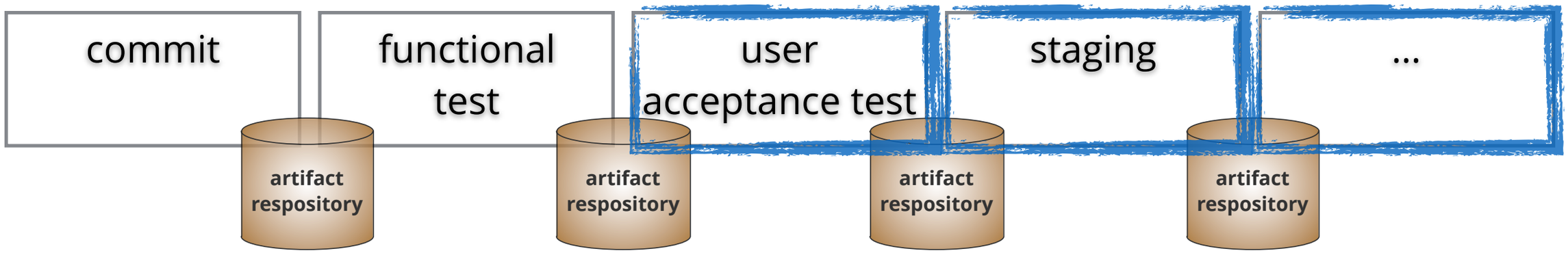


Run against each check-in (continual integration)

Starts building a release candidate

If it fails, fix it immediately

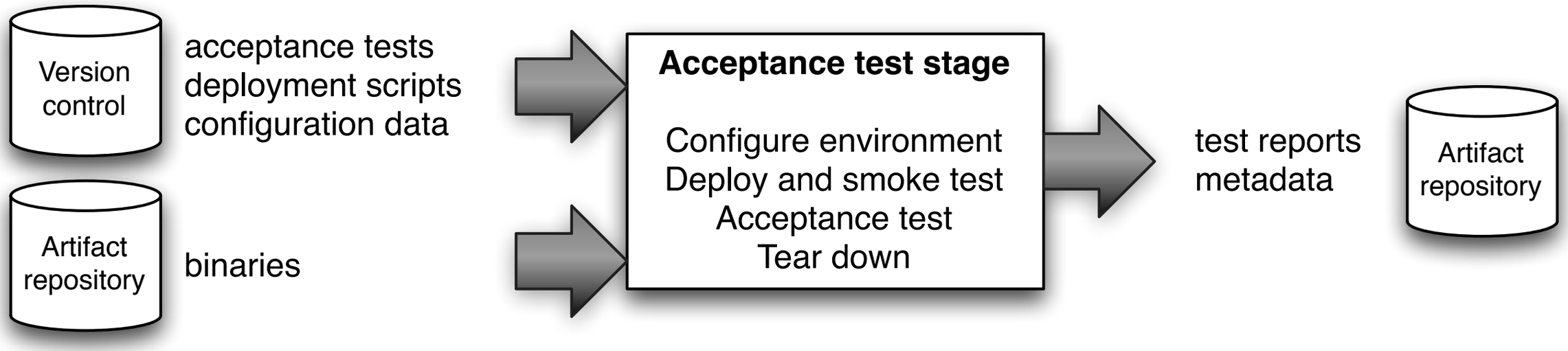
Pipeline Construction



increasing confidence in production readiness

Pipeline stages = feedback opportunities

UAT Stage

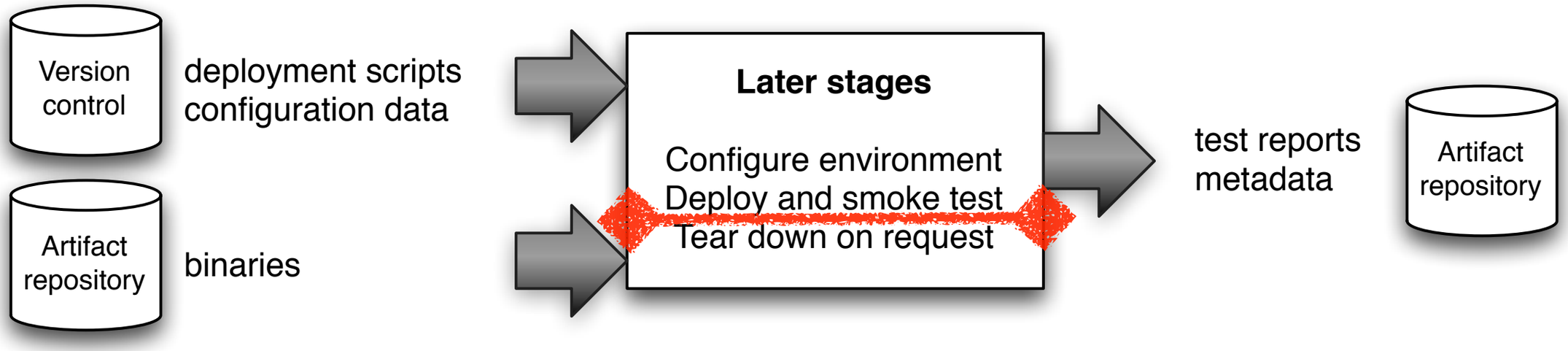


End-to-end tests in production-like environment

Triggered when upstream stage passes

First DevOps-centric build

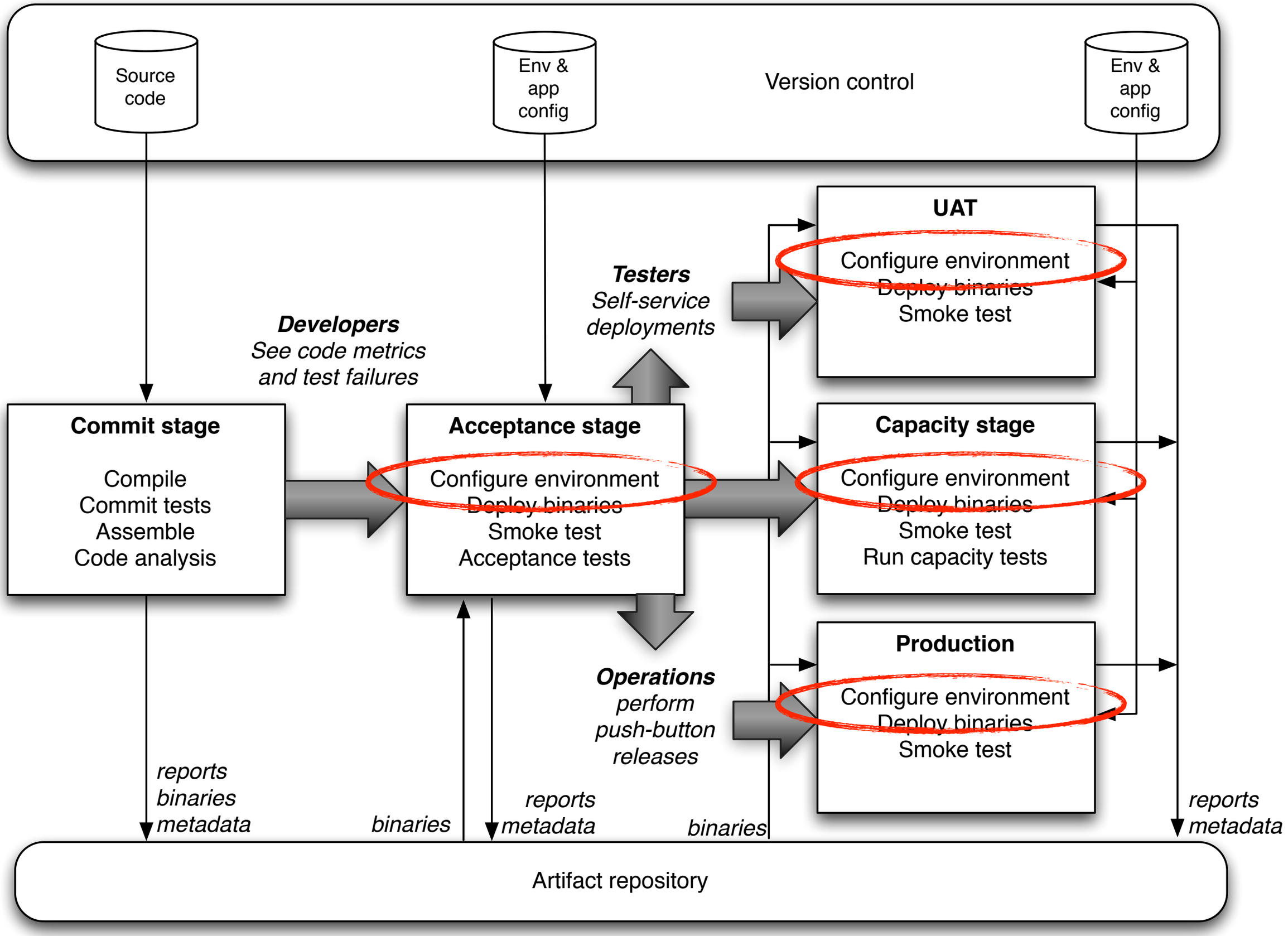
Manual Stage



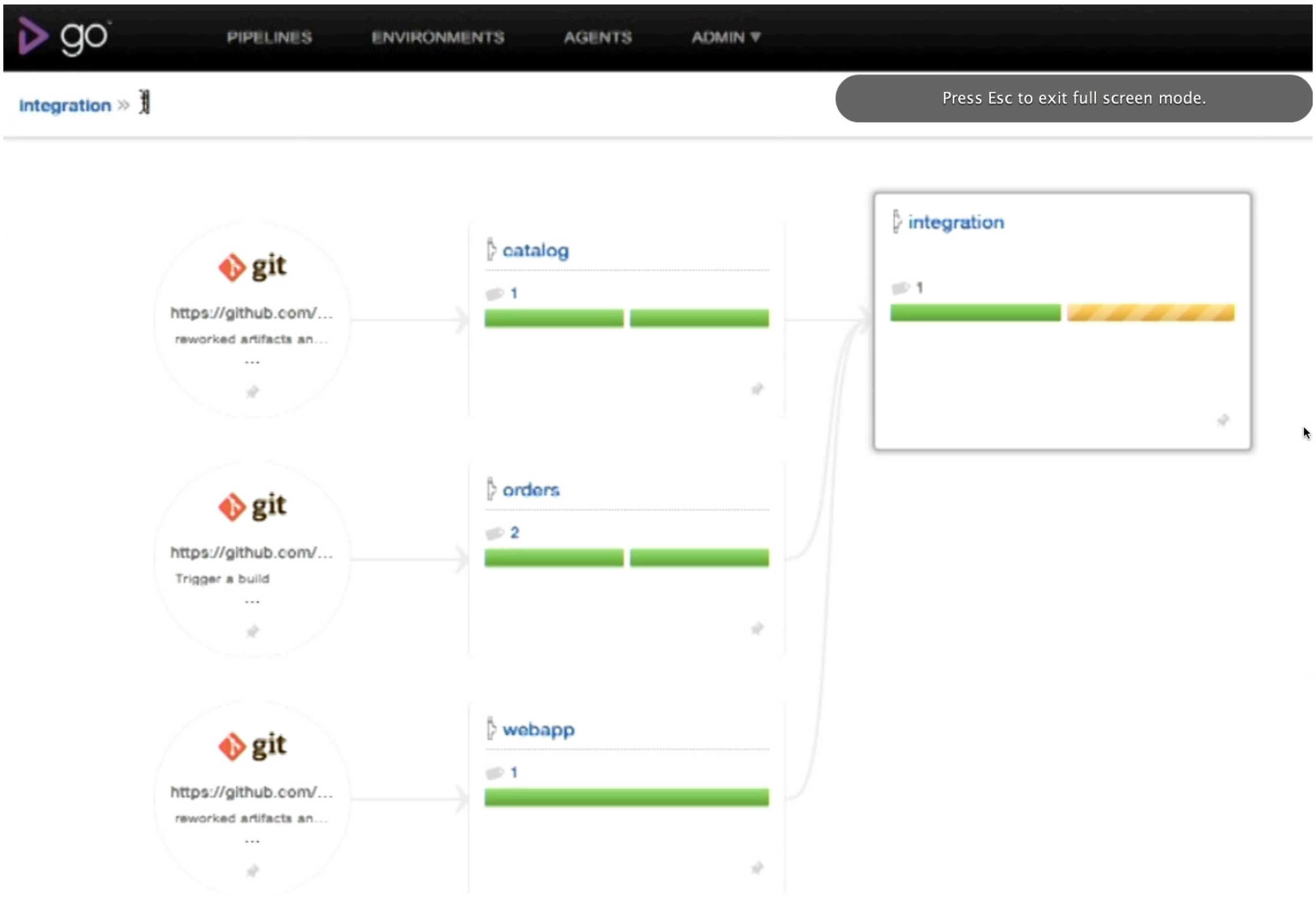
UAT, staging, integration, production, …

Push versus Pull model

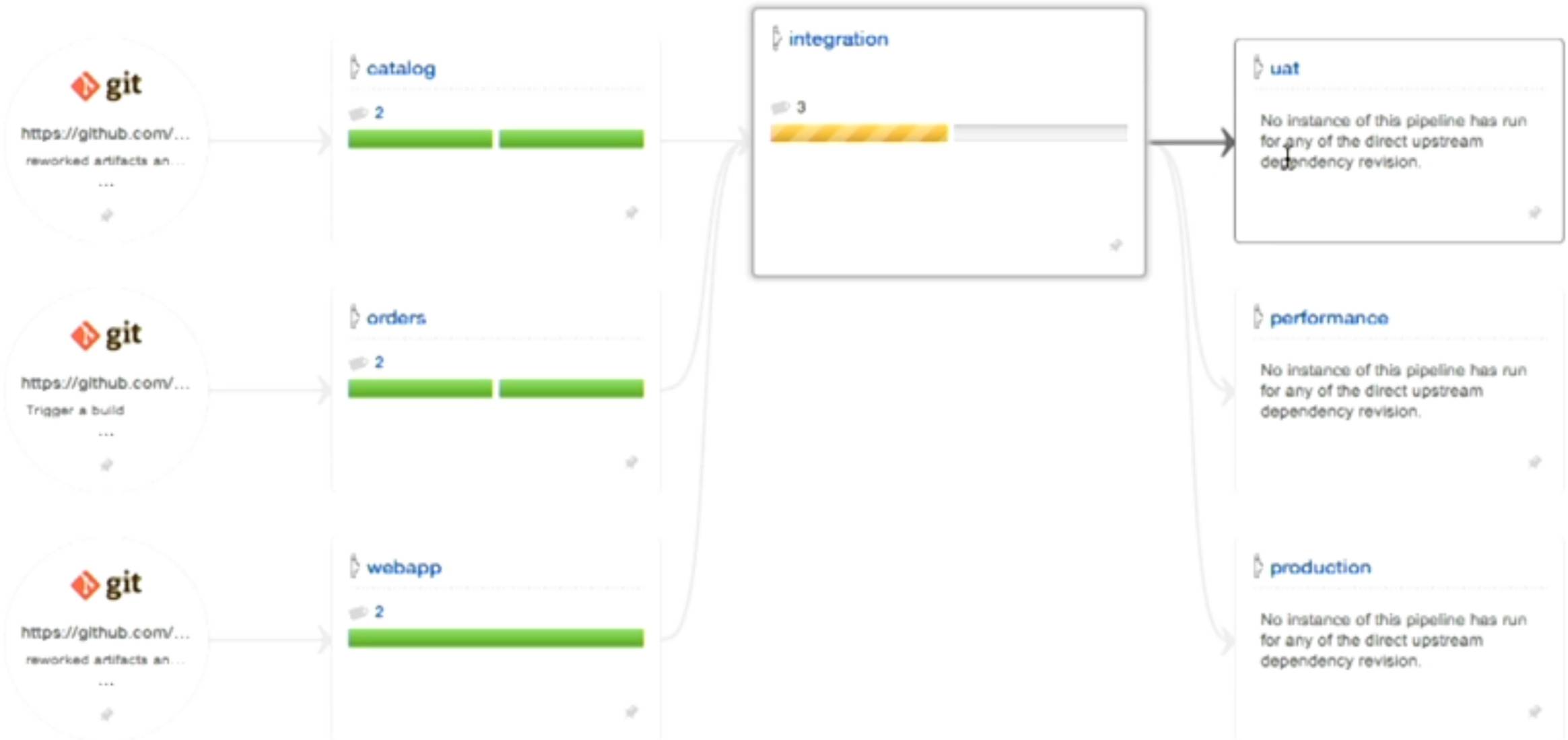
Deployments self-serviced through push-button process


Source code
Env & app config
Version control
Env & app config
UAT
Configure environment
Deploy binaries
Smoke test
Testers
Self-service deployments
Developers
See code metrics and test failures
Commit stage
Compile
Commit tests
Assemble
Code analysis
Acceptance stage
Configure environment
Deploy binaries
Smoke test
Acceptance tests
Capacity stage
Configure environment
Deploy binaries
Smoke test
Run capacity tests
Production
Configure environment
Deploy binaries
Smoke test
Operations
perform push-button releases
reports
binaries
metadata
binaries
reports
metadata
binaries
reports
metadata
Artifact repository

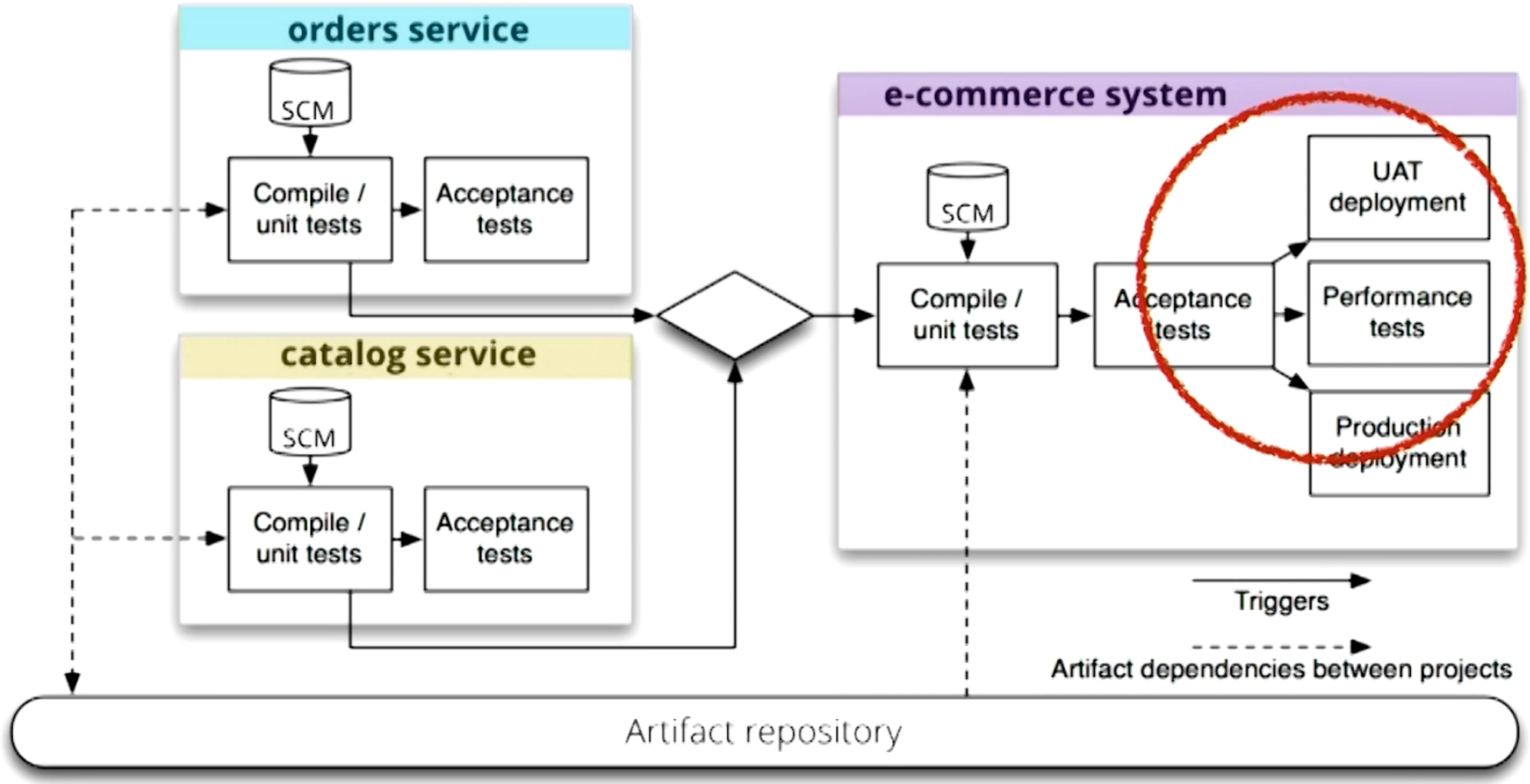
Integration Pipeline in Go CD



Integration Pipeline in Go CD

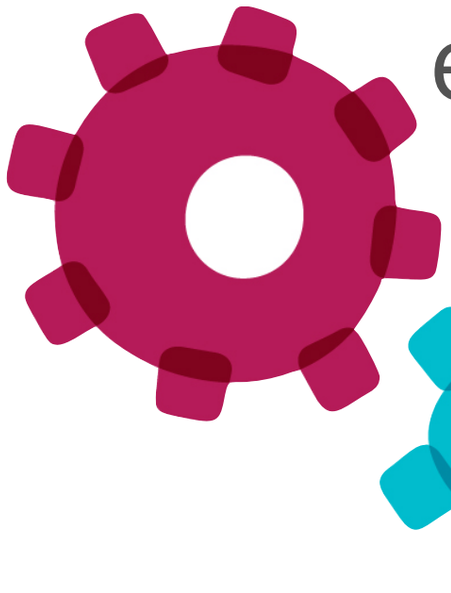


Deployment Pipeline Fan Out



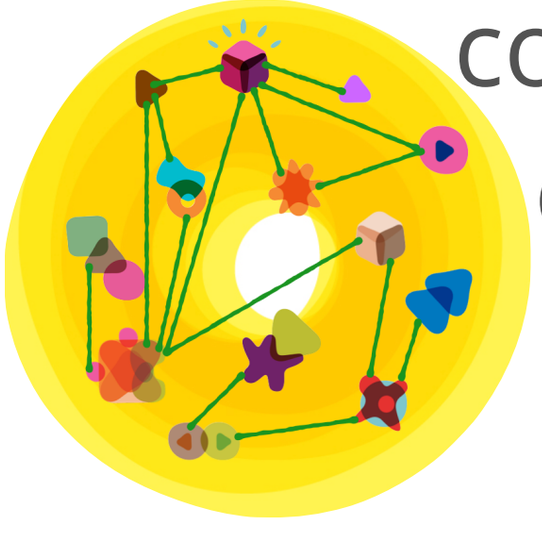
Deployment Pipeline Fan Out

Agenda



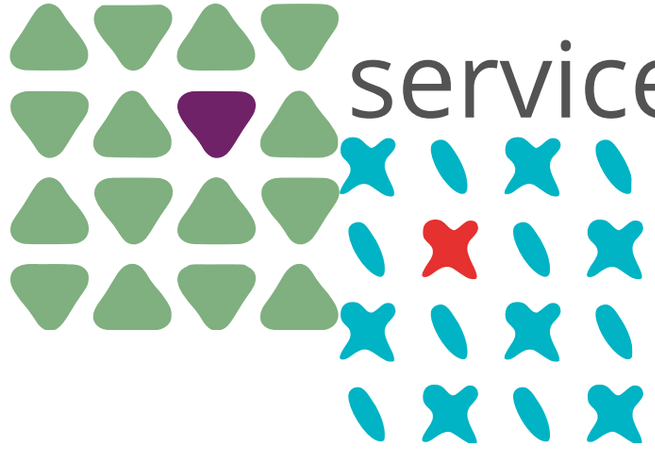
engineering practices

deployment models

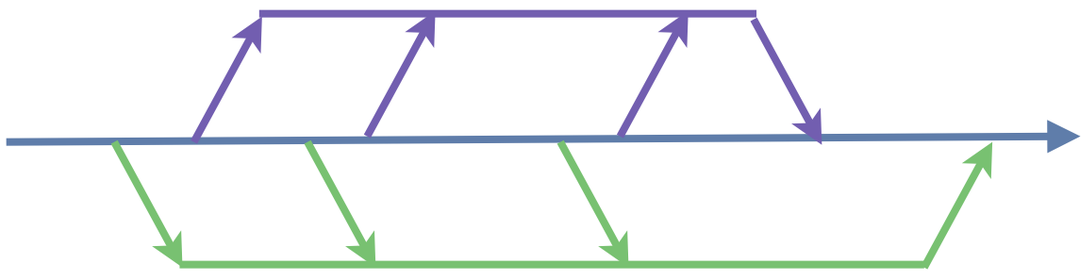


coupling & cohesion

components & services



Feature Branching



No Refactoring

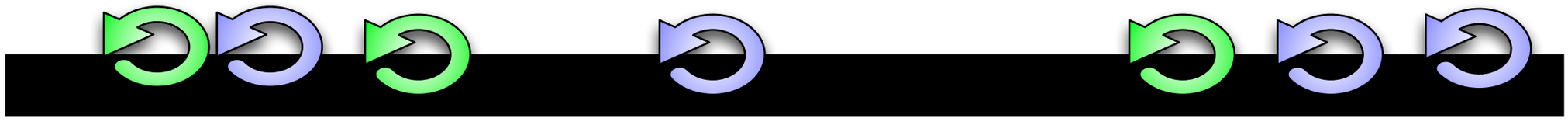
No combined features

Big Scary Merge

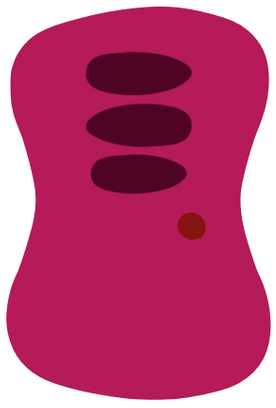
Feature Branching

versus

Trunk-based Development



trunk-based development



feature toggles

Config File

```
[featureToggles]
wobblyFoobars: true
flightyForkHandles: false
```

some.jsp

```
<toggle name=wobblyFoobars>
  ... various UI elements
</toggle>
```

other.java

```
forkHandle = (featureConfig.isOn(‘flightlyForkHandles)) ?
                new FlightyForkHander(aCandle) :
                new ForkHandler(aCandle)
```

Togglz - Features flag for Java

www.togglz.org

Reader

Feature Flags for the Java plattform

MAIN

- Home
- Downloads
- Source Code
- Forums
- Issue Tracker
- stackoverflow.com
- Continuous Integration
- License

REFERENCE

- What's new?
- Getting Started
- Javadocs 2.0.0.Final
- Javadocs 1.1.0.Final
- Javadocs 1.0.0.Final
- Updating Notes

DOCUMENTATION

Togglz

What is it about?

Togglz is an implementation of the Feature Toggles pattern for Java. Feature Toggles are a very common agile development practices in the context of continuous deployment and delivery. The basic idea is to associate a toggle with each new feature you are working on. This allows you to enable or disable these features at application runtime, even for individual users.

Want to learn more? Have a look at an usage example or check the quickstart guide.

News

01-Jul-2013

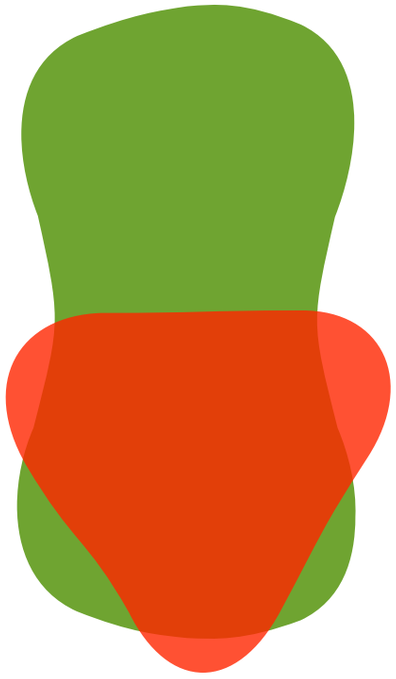
Togglz 2.0.0.Final released

I'm very happy to announce the release of Togglz 2.0.0.Final. This new version is the result of many months of hard work. Many core concepts of Togglz have been revised to provide much more flexibility.

The most noteworthy change in Togglz 2.0.0.Final is the new extendible feature activation mechanism that allows to implement custom strategies for activating features. Beside that there are many other updates.

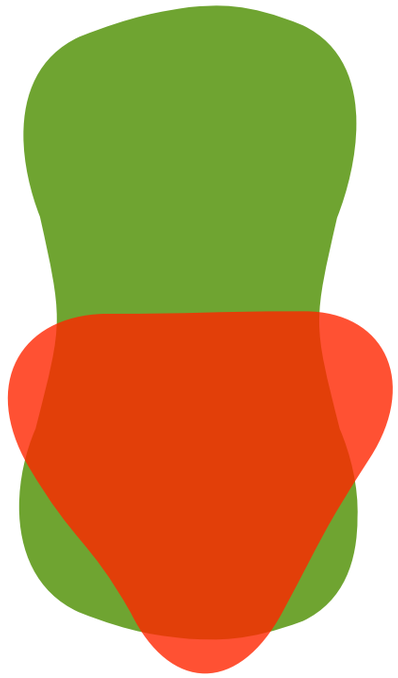
www.togglz.org

```java
@FeatureGroup
@Label("Performance Improvements")
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Performance {
   // no content
}
```



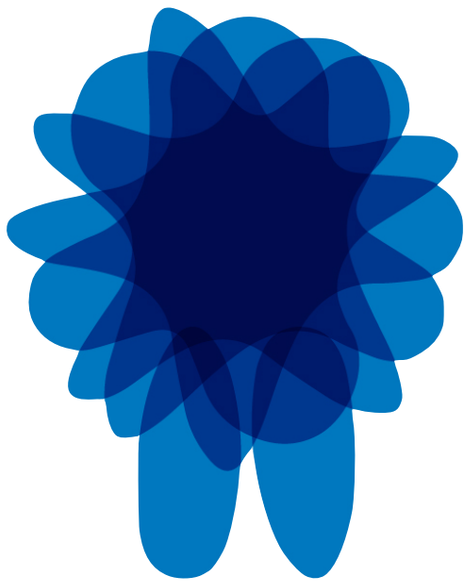
removed as soon as feature decision is resolved

Feature toggles are purposeful technical debt added to support engineering practices like Continuous Delivery.

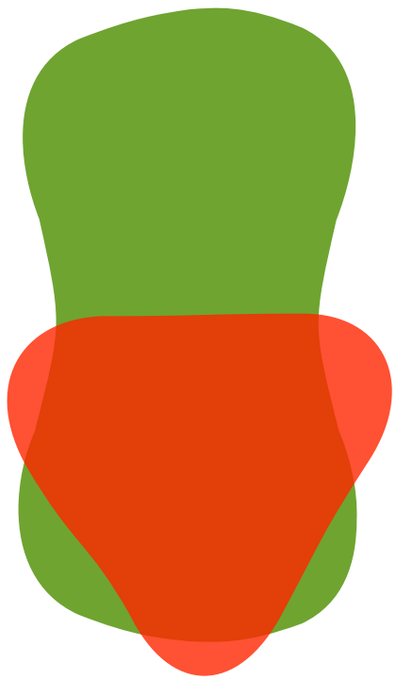


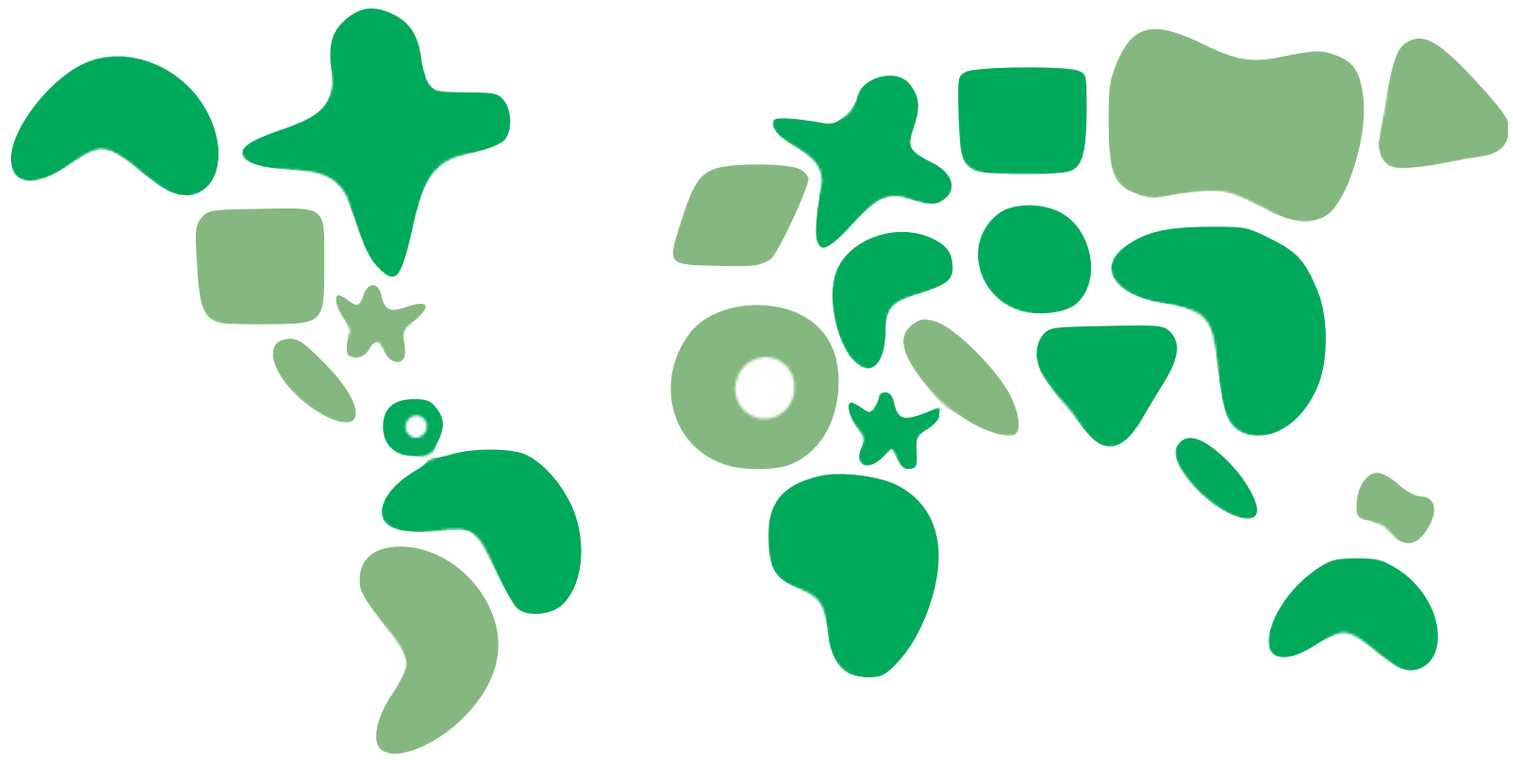
taxonomy

```
@FeatureGroup
@Label("Performance Improvements")
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Performance {
  // no content
}
```

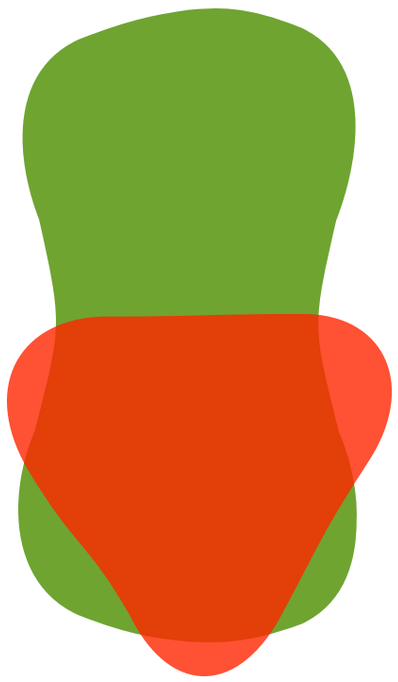


works on all platforms &
technology stacks

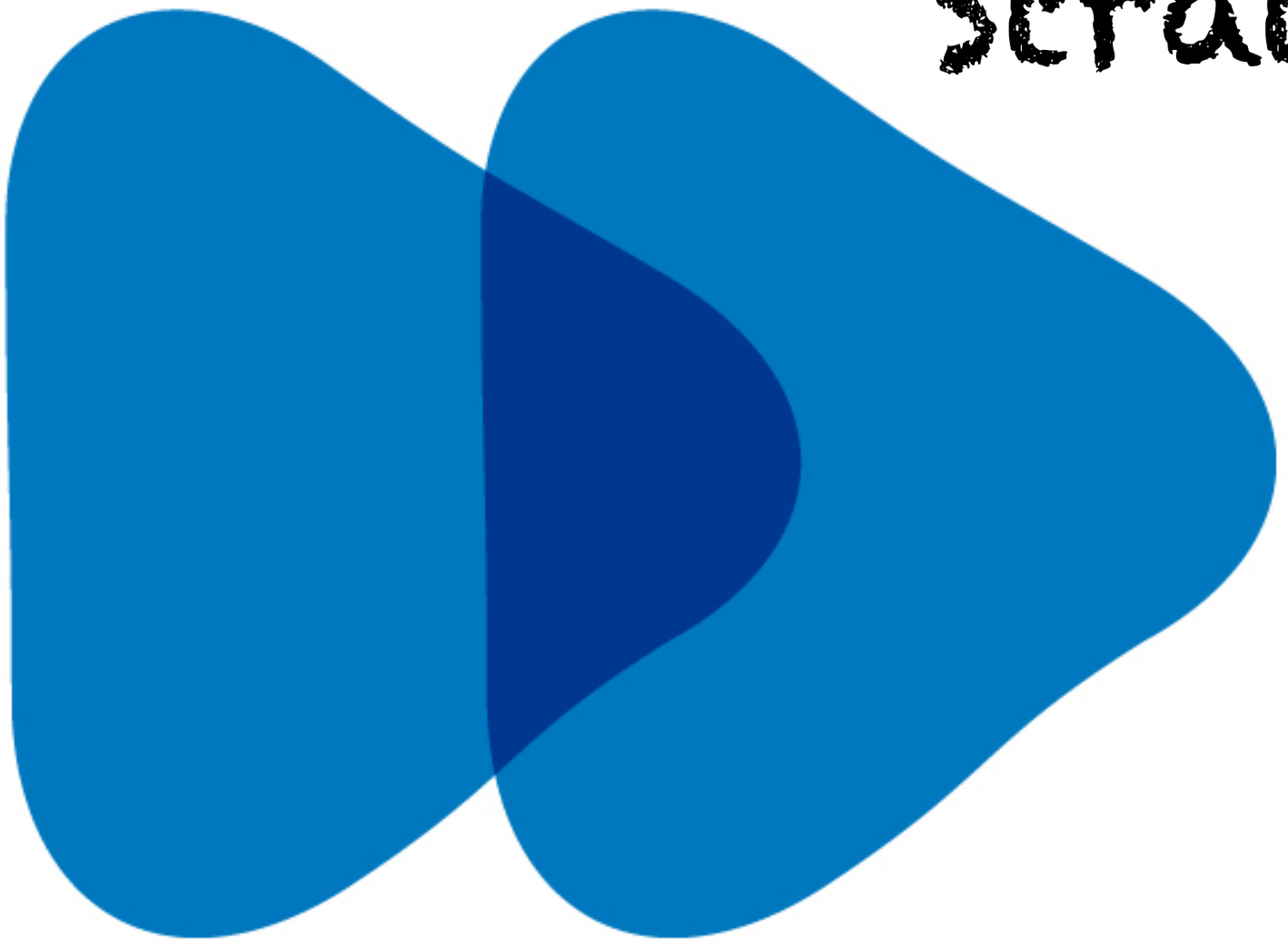




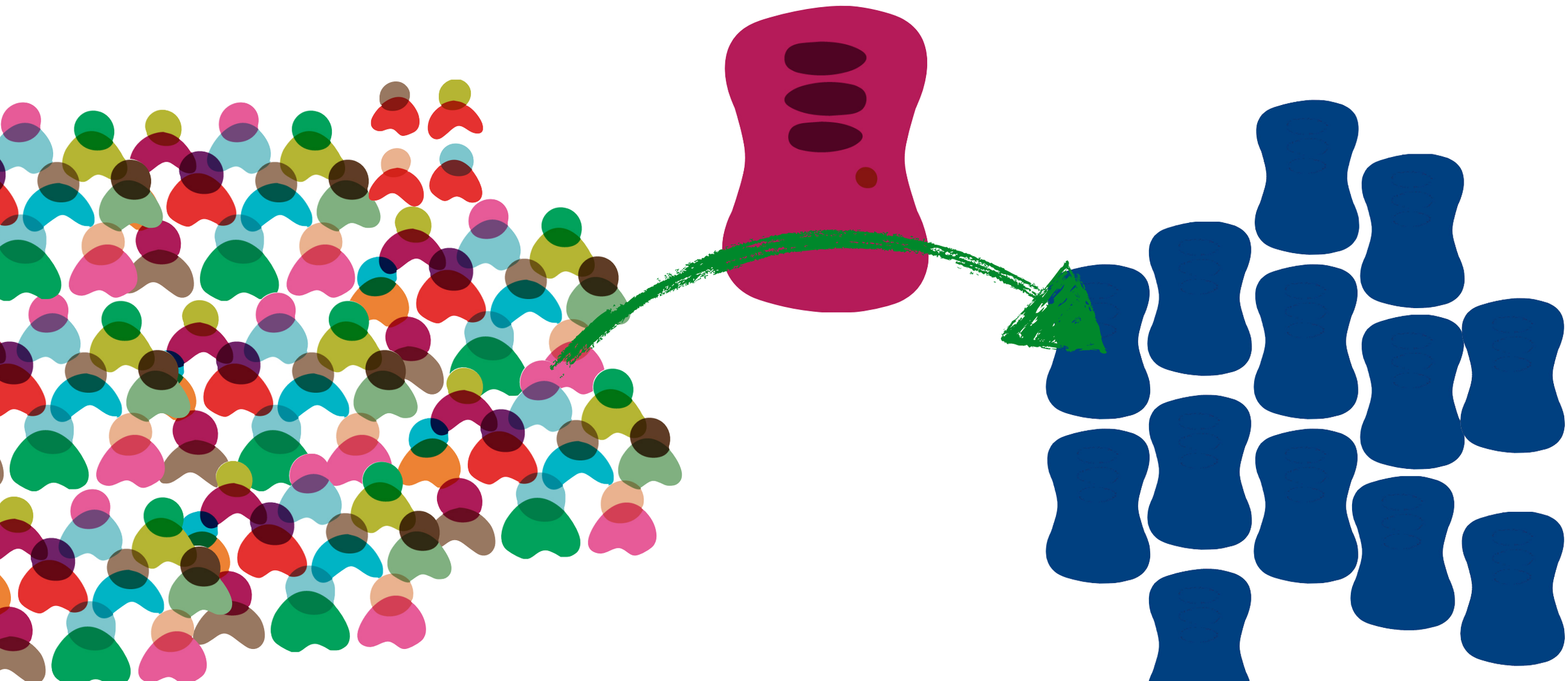
ubiquitous



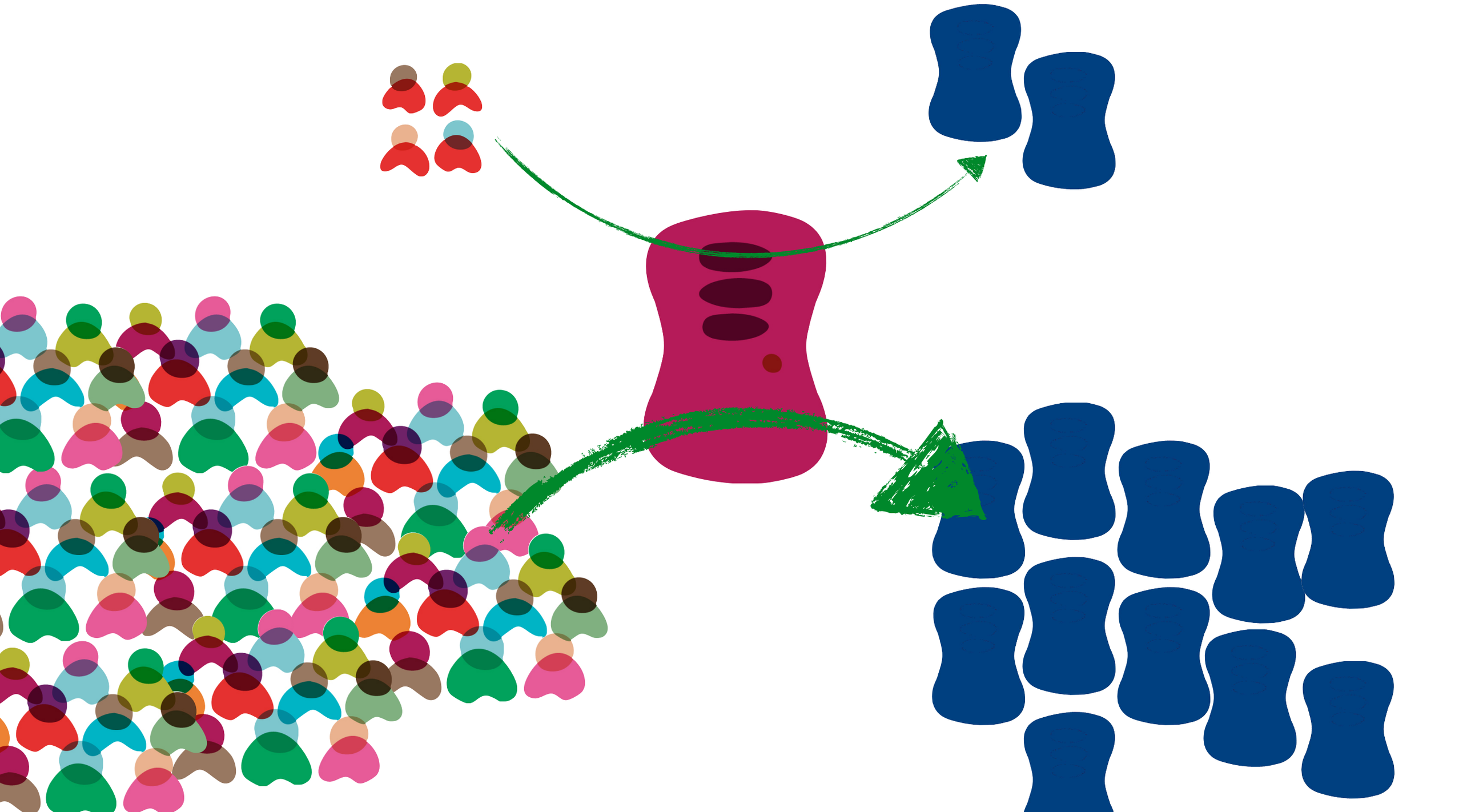
Release Strategies



Canary Releasing

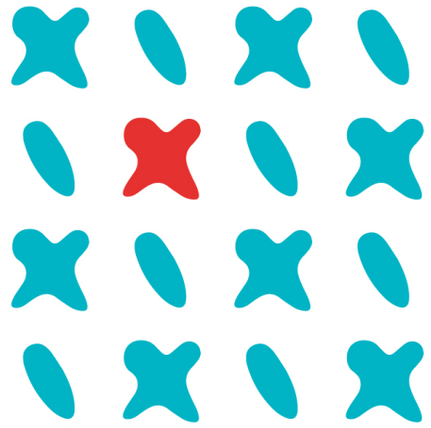


Canary Releasing

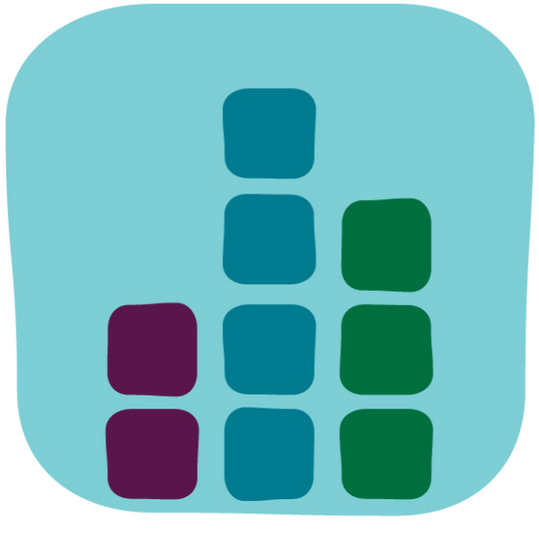


Canary Releasing

reduce risk of release



multi-variant testing



performance testing

Togglz - Features flag for Java

www.togglz.org/documentation/activation-strategies.html

Reader

Feature Flags for the Java platform

MAIN

- Home
- Downloads
- Source Code
- Forums
- Issue Tracker
- stackoverflow.com
- Continuous Integration
- License

REFERENCE

- What's new?
- Getting Started
- Javadocs 2.0.0.Final
- Javadocs 1.1.0.Final
- Javadocs 1.0.0.Final
- Updating Notes

DOCUMENTATION

- Overview
- Installation
- Configuration
- Usage
- Admin Console

Activation Strategies

Togglz defines the concept of *activation strategies*. They are responsible to decide whether an enabled feature is active or not. Activation strategies can for example be used to activate features only for specific users, for specific client IPs or at a specified time.

Togglz ships with the following default strategies:

- Username
- Gradual rollout
- Release date
- Client IP
- Server IP
- ScriptEngine

The following sections will describe each strategy in detail. The last section custom strategies describes how to build you own strategies.

Username

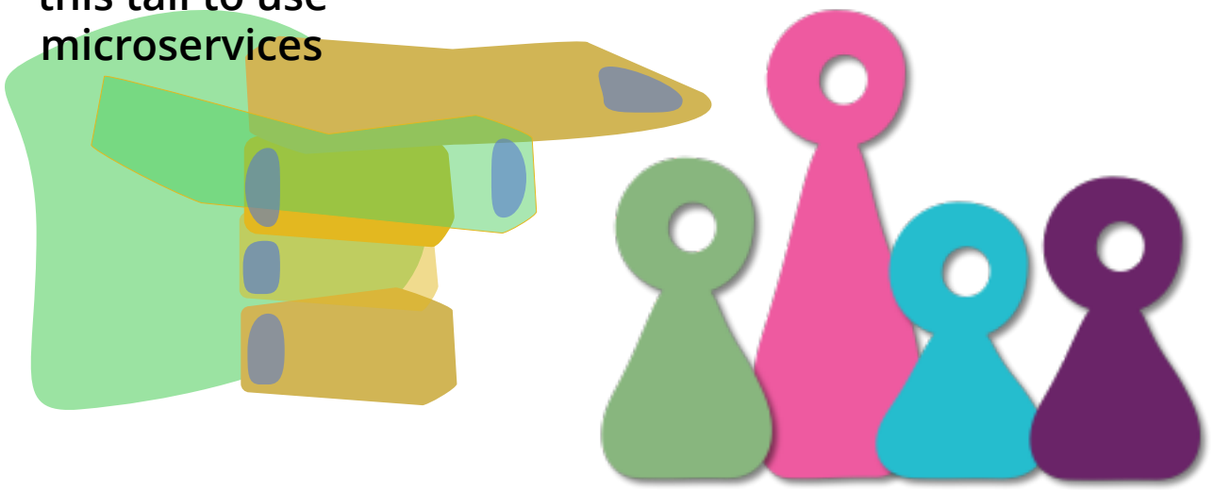
Enabling features for specific users was already supported in very early versions of Togglz, even before the activation strategy concept was introduced in Togglz 2.0.0.

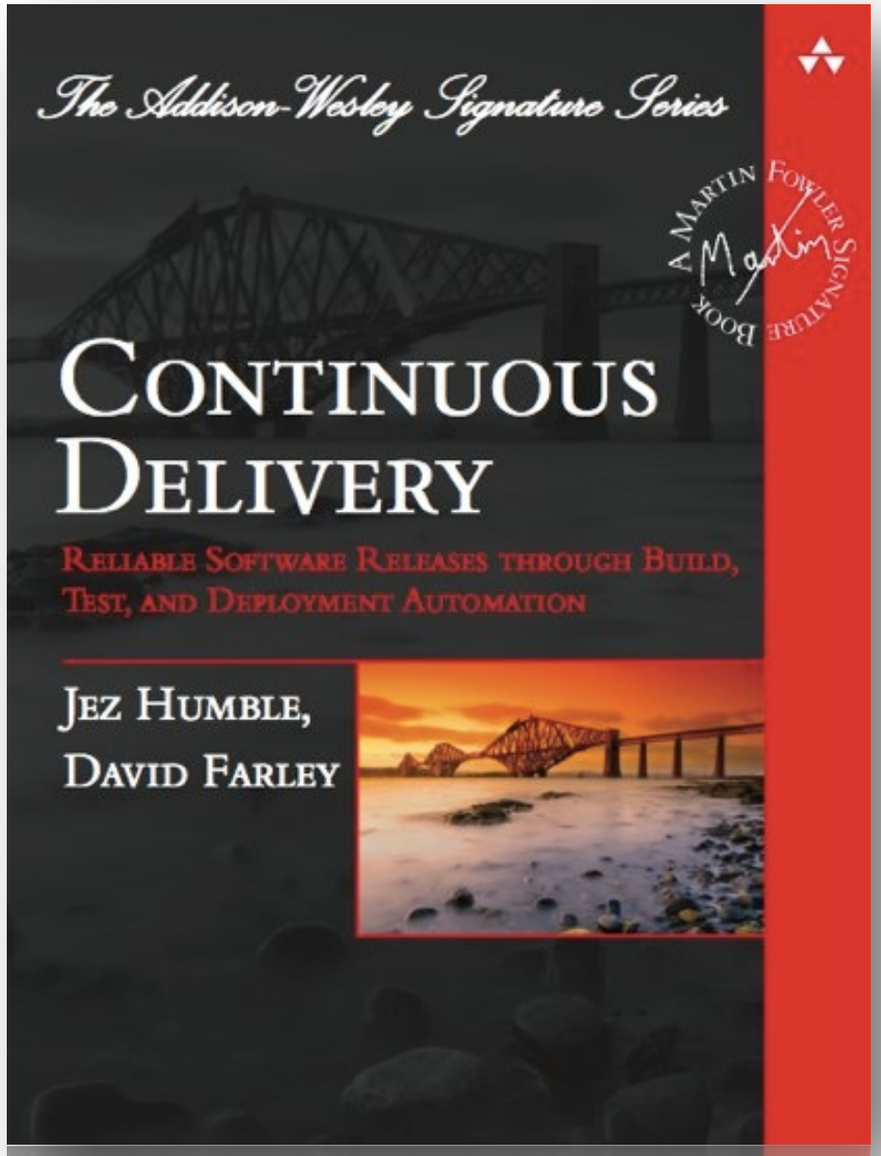
If you select this strategy for a feature, you can specify an comma-separated list of users for which the feature should be active. Togglz will use the `UserProvider` you configured for the FeatureManager to determine the current user and compare it to that list.

Please note that Togglz will take case into account when comparing the usernames. So the users `admin` and `Admin` are NOT the same.

www.togglz.org

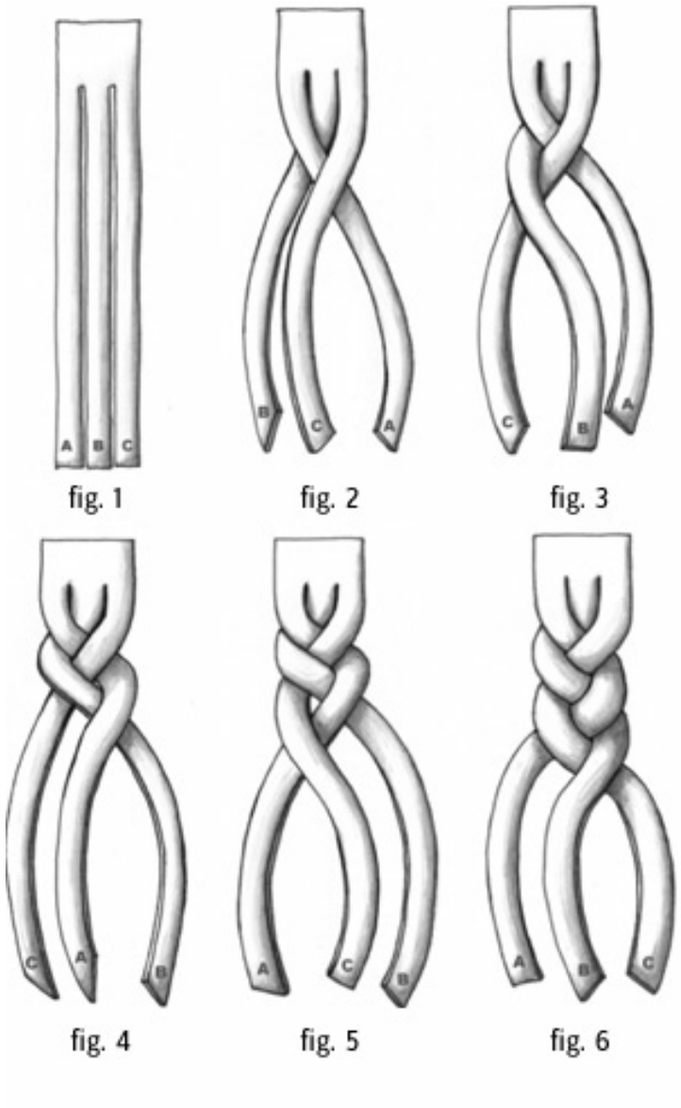
You must be this tall to use microservices



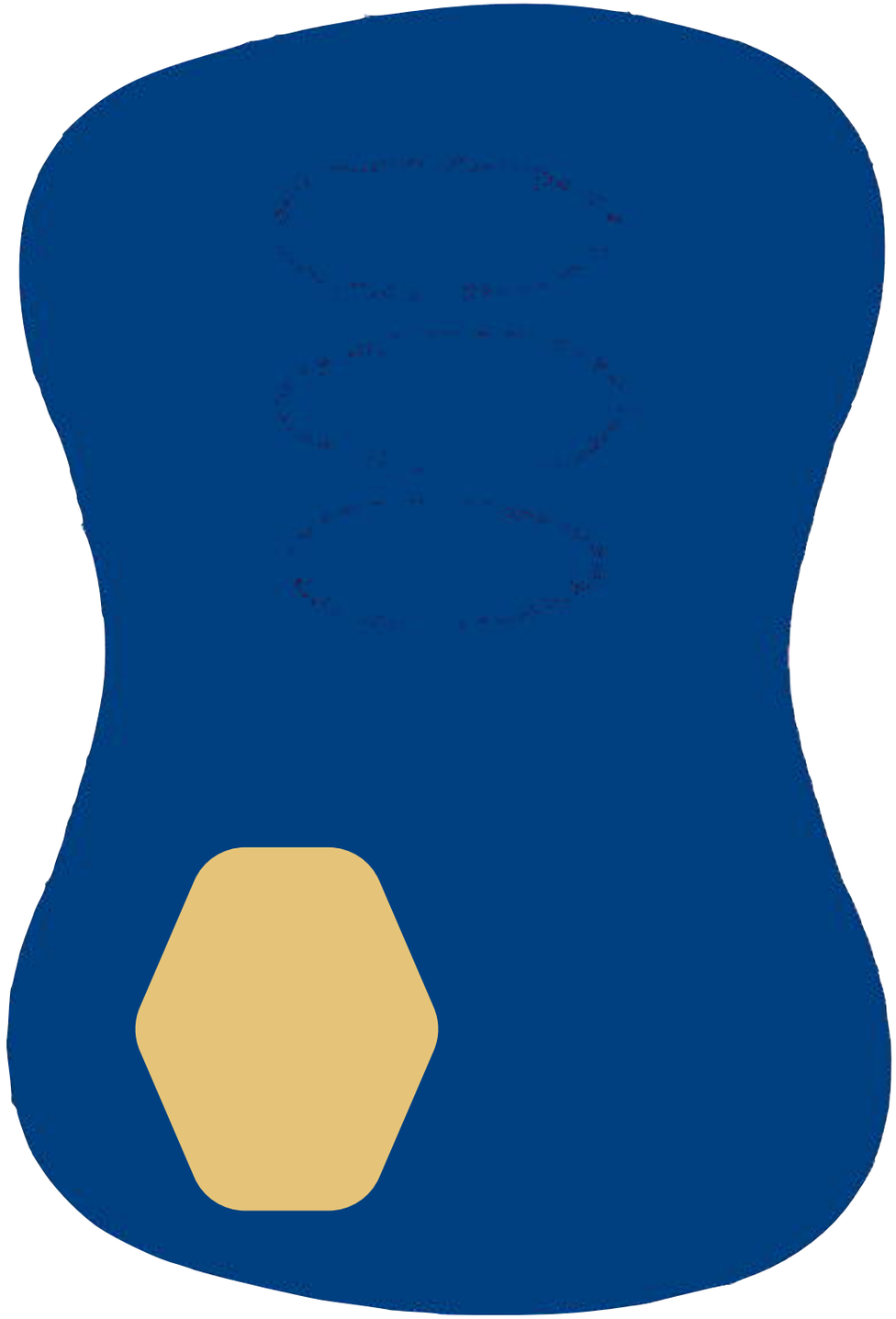


Mature engineering practices.

Complected Deployments



complect, *transitive verb*:
intertwine, embrace, especially
to plait together



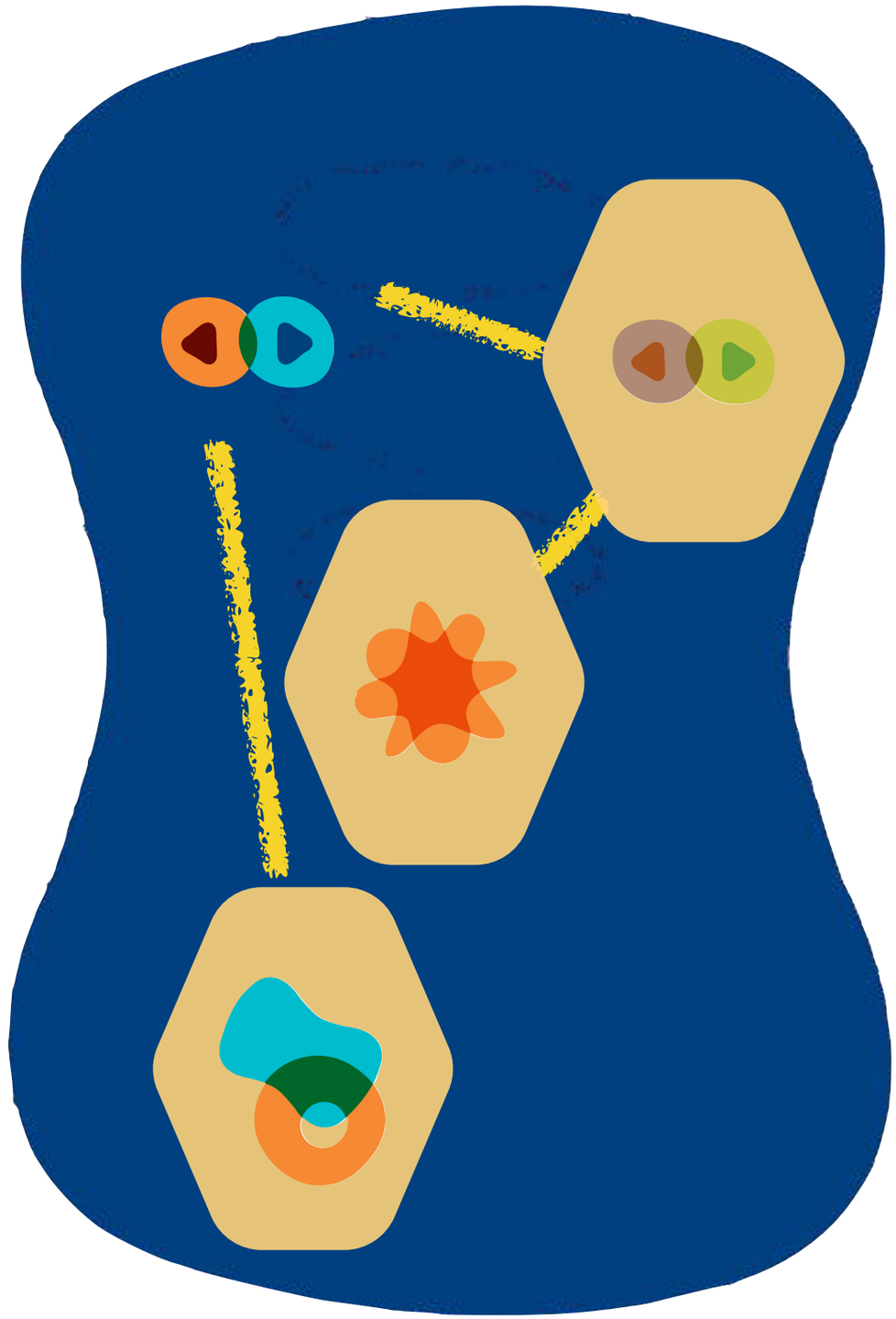
production

Evolutionary Architecture

Components are *deployed*.

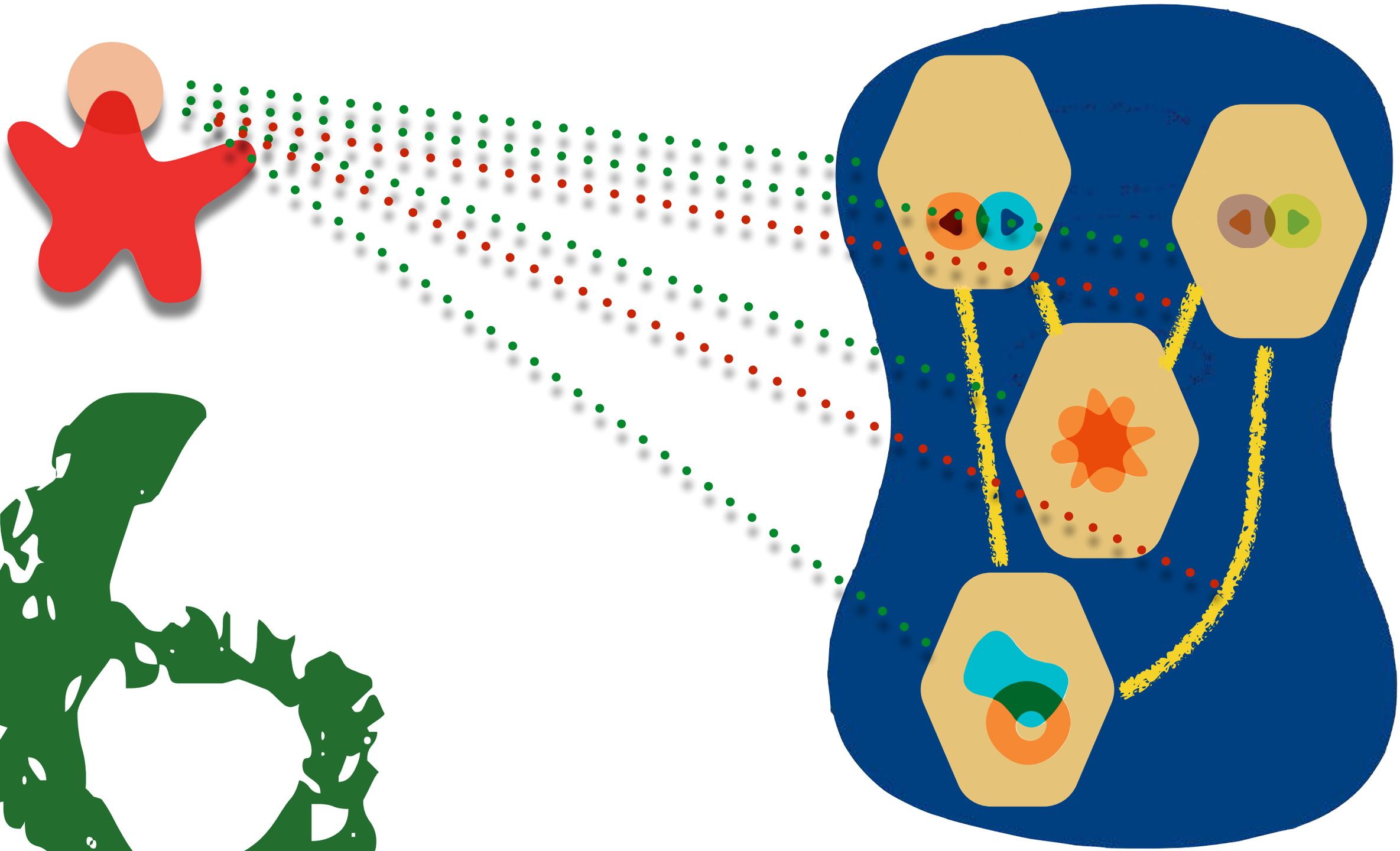
Features are *released*.

Applications consist of *routing*.



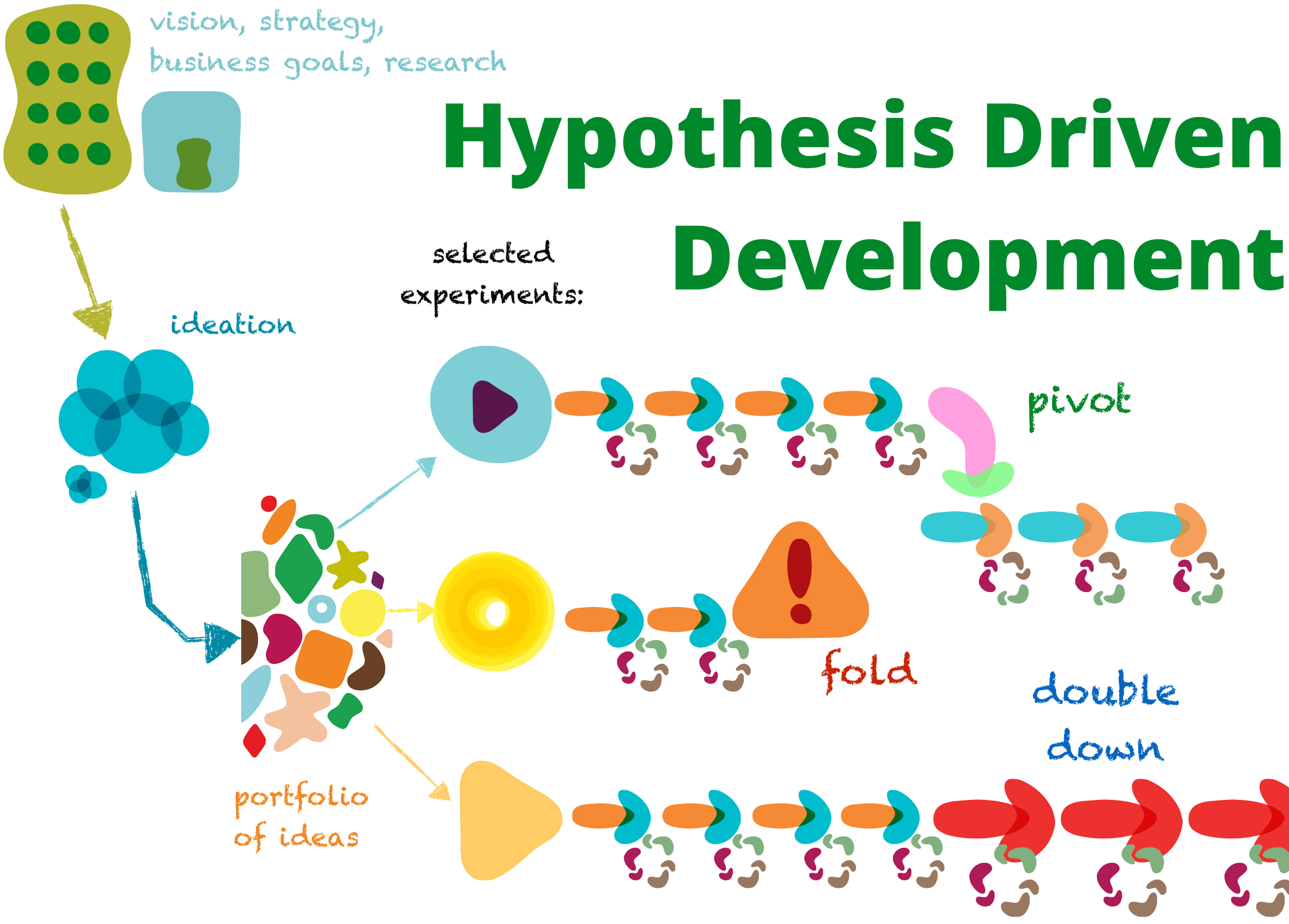
production

Towards Evolutionary Architecture



production

Hypothesis Driven Development

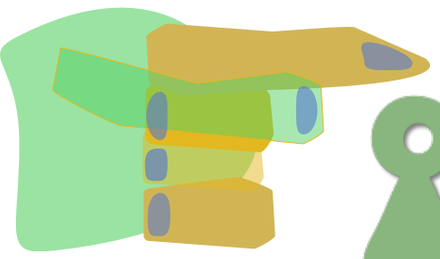


Continuous Delivery for Architects

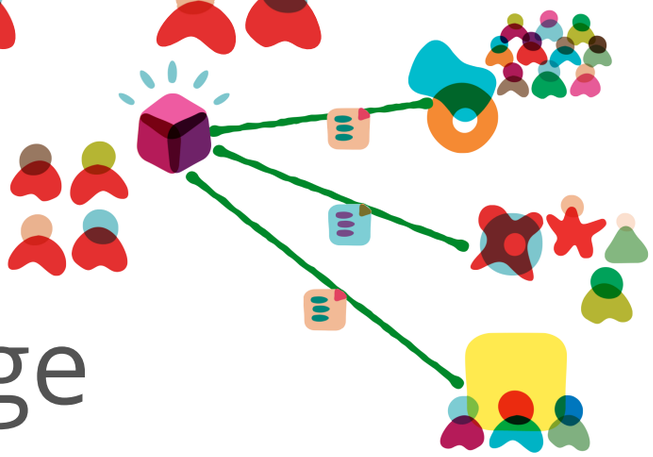
Yesterday's best practice is tomorrow's antipattern.

Understanding shifting structure.

Expanding role of architect.

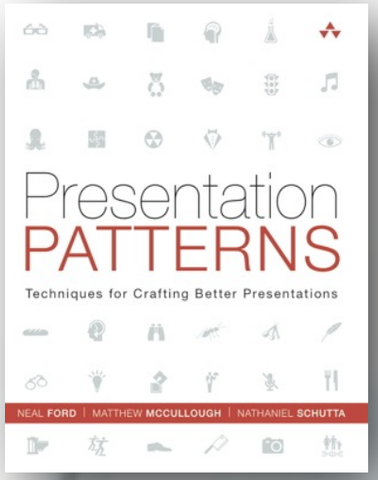


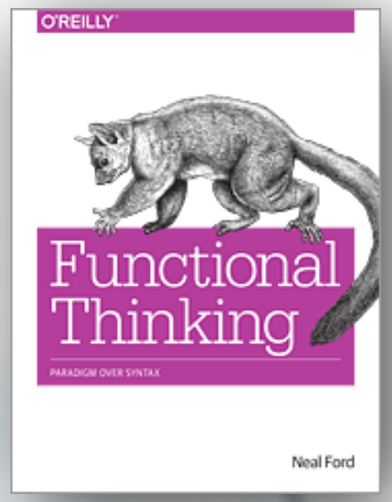
Mature engineering practices.



Manage coupling intelligently.

towards evolutionary architecture...





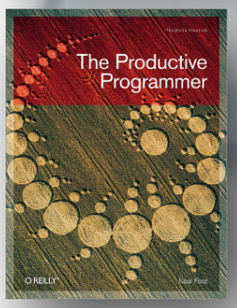
nealford.com

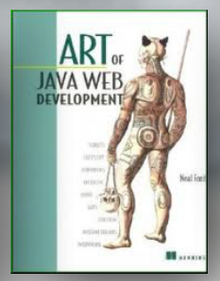
@neal4d

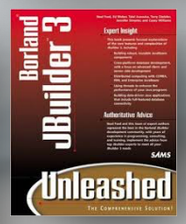
ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler

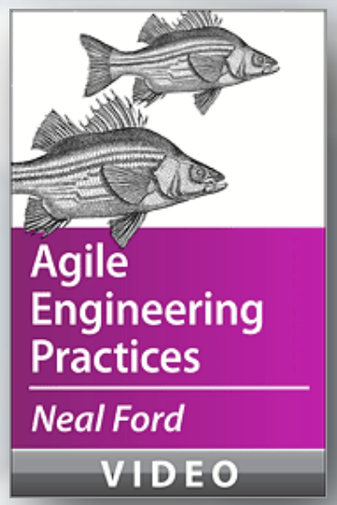


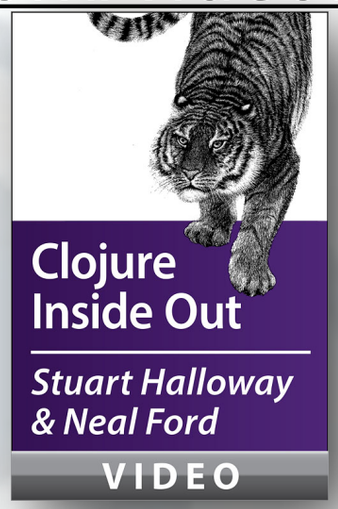


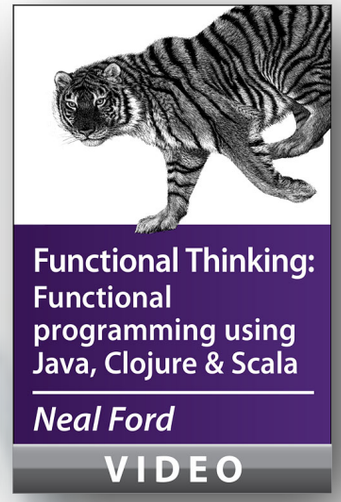


nealford.com/books

nealford.com/videos







www.oreilly.com/software-architecture-video-training-series.html